

AD-A070 093

NAVAL POSTGRADUATE SCHOOL MONTEREY CA

F/G 9/2

ATHENA: A SYSTEM TO INTERACTIVELY ANALYZE LARGE SCALE OPTIMIZAT--ETC(U)

MAR 79 P I GALATAS

UNCLASSIFIED

OF 2

AD  
A070093



NL



TESTED

1 OF 2

AD  
A070093



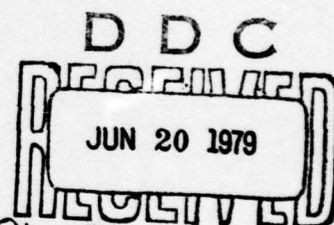


LEVEL *H*

NAVAL POSTGRADUATE SCHOOL  
Monterey, California

*2*

AD A 070093



THESIS

DDC FILE COPY

ATHENA: A SYSTEM TO INTERACTIVELY ANALYZE  
LARGE SCALE OPTIMIZATION MODELS

by

Panagiotis Ioannou Galatas

March 1979

Thesis Advisor:

G. Bradley

Approved for public release; distribution unlimited.

79 06 20 064

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle)		5. TYPE OF REPORT & PERIOD COVERED
(6) ATHENA: A System to Interactively Analyze Large Scale Optimization Models		Master's Thesis, March 1979
7. AUTHOR(s)		6. PERFORMING ORG. REPORT NUMBER
(10) Panagiotis Ioannou/Galatas		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
		(12) 700p.
11. CONTROLLING OFFICE NAME AND ADDRESS		11. REPORT DATE
Naval Postgraduate School Monterey, California 93940		March 1979
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		13. NUMBER OF PAGES
Naval Postgraduate School Monterey, California 93940		101
		15. SECURITY CLASS. (of this report)
		Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)		
Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Optimization Model		
79 06 20 064		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		
<p>The analysis of the solutions of large scale optimization models is very difficult without computer aids because typical outputs may exceed 40,000 lines. A computer system to allow efficient storage and interactive analysis of the solution file was designed and implemented in FORTRAN. This</p>		

DD FORM 1473  
1 JAN 73  
(Page 1)EDITION OF 1 NOV 68 IS OBSOLETE  
S/N 0102-014-6601

UNCLASSIFIED

1

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251 4/50

JCB



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE/When Data Entered

(20. ABSTRACT Continued)

system, called ATHENA, has been initially designed for problems with up to 30,000 rows and columns; tests show the system has fast response time for these large problems. A description of the system, its data structures and commands as well as a user's manual is included.

Accession For	
NTIS GRI&I	<input checked="checked" type="checkbox"/>
DDC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/ _____	
Availability Codes	
Dist	Avail and/or special
A	

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE/When Data Entered

Approved for public release; distribution unlimited.

ATHENA: A System to Interactively Analyze  
Large Scale Optimization Models

by

Panagiotis Ioannou Galatas  
Captain, Hellenic Army  
B.S., Military Academy of Greece, 1965  
M.S., Highest School of Economic Sciences of Athens, 1976

Submitted in partial fulfillment of the  
requirements for the degrees of

MASTER OF SCIENCE IN OPERATIONS RESEARCH  
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

March 1979

Author

Approved by:

Thesis Advisor

GERALD G. BROWN

Second Reader

Chairman, Department of Operations Research

Chairman, Department of Computer Science

Dean of Information and Policy Science

# ABSTRACT

↓ The analysis of the solutions of large scale optimization models is very difficult without computer aids because typical outputs may exceed 40,000 lines. A computer system to allow efficient storage and interactive analysis of the solution file was designed and implemented in FORTRAN. This system, called ATHENA, has been initially designed for problems with up to 30,000 rows and columns; tests show the system has fast response time for these large problems. A description of the system, its data structures and commands as well as a user's manual is included. ↗



## TABLE OF CONTENTS

I.	INTRODUCTION -----	8
II.	SOFTWARE DESCRIPTION -----	12
	A. GENERAL -----	12
	1. Parser - Code Generator -----	12
	2. Input -----	12
	3. Interpreter -----	12
	B. LANGUAGE -----	12
	1. Host Language -----	12
	2. Query Language -----	14
	C. DATA STRUCTURES -----	15
	1. Preview -----	15
	2. SPARSE Data Structure -----	17
	3. SUPERSPARSE Data Structure -----	22
III.	USER'S MANUAL -----	24
	A. INTRODUCTION -----	24
	B. QUERY LANGUAGE -----	24
	1. Control Queries -----	25
	a. VERIFY -----	25
	b. NOVERIFY -----	25
	c. PROMPT -----	25
	d. NOPROMPT -----	26
	e. H Any String -----	26
	f. * Any String -----	26
	g. END -----	26

2. Command Queries -----	26
a. TYPE Field -----	27
b. SELECT Field -----	29
c. MASK Field -----	30
d. CONDITION Field -----	31
e. PRINT OPTION Field -----	34
3. The SET Command -----	35
C. ERROR MESSAGES -----	37
D. LIMITATIONS - EXTENSIONS -----	40
E. SYSTEM INTERFACE -----	42
1. Preview -----	42
2. Input File -----	43
3. Packed File -----	44
F. EXAMPLE OF SYSTEM USE -----	45
1. Obtaining the Unpacked File -----	45
2. Using the System under CP/CMS -----	46
APPENDIX: COMPUTER PROGRAM LISTING -----	54
LIST OF REFERENCES -----	100
INITIAL DISTRIBUTION LIST -----	101

### ACKNOWLEDGMENT

The author would like to acknowledge the contributions of Professor Gordon Bradley whose mathematical programming expertise was of invaluable assistance in analyzing this problem, Professor Gerald Brown for his valuable guidance and the author's wife Lina Galatas who provided the encouragement and secretarial skills necessary for successful completion. The system name ATHENA has been chosen for several reasons. First, ATHENA is the goddess of wisdom from ancient Greece. Second, it is the Greek name of the capital of my country. Finally, ATHENA is my mother's name.



## I. INTRODUCTION

The generation, solution and analysis of large scale mathematical programming models presents significant problems in data handling. For a typical large scale model the output from the computer may exceed 40,000 lines and it is very difficult and time consuming for the user to go through that much paper to extract the information he needs. Space is also needed to store all these outputs and it is difficult to provide routine access to old solution files.

Over the past several years, faculty and students at the Naval Postgraduate School and the University of California at Los Angeles have been cooperatively developing theory and algorithms to solve large scale linear, nonlinear and integer optimization problems. This research has been furthered by the development of several software systems to solve large scale optimization problems. ATHENA is part of that development and was built to satisfy a pressing need to be able to quickly and easily analyze solutions to these large problems.

The research in large scale optimization at the Naval Postgraduate School has concentrated on providing economic solutions to current Department of Defense problems. One such project that was going on concurrent with the development of ATHENA was a large, medium range capital budgeting problem that required the solution of a mixed integer programming

problem with 11,687 constraints and variables [6]. ATHENA was used successfully with this project, and performance of ATHENA on this problem is reported below.

System ATENA has been developed to handle output from these large optimization problems by enabling the user to get the information he needs interactively through a computer terminal and by economically storing the large files in packed form in low cost media. The features of the system may be summarized as follows:

1. Quick and accurate answers to simple questions that are tedious and error prone to address manually, e.g.,

How many of a set of variables are = 1.0?

or       How many are greater than 0?

or       What variables are in a specified range?

or       What constraints are satisfied exactly? Etc.

2. In large scale mathematical programming models the names of rows and columns are constructed systematically so that groups of variables with relationships in the real world have similar names. Using the system one can have automatic, easy and accurate answers for many interesting properties of these groups. (For instance, the average value of all the variables whose names begin with X, etc.)

3. ATHENA can also be used as a basis for a simple report writer.

4. The system provides very compact computer storage: the original solution file is typically packed into 1/10 of its normal volume. For example, a solution file from the

IBM MPS/360 package of a linear programming model with 12,000 rows and columns occupies 1.5 magnetic tapes 2400 feet long at 800 BPI in original unblocked form, but in packed form the solution occupies approximately 31 feet of magnetic tape.

5. There is systematic and economic file organization with easy and accurate access.

6. A file structure for multiple runs of the problem for comparisons is available.

7. The system requires modest resources (core, time) in a time sharing environment for a large amount of data.

8. The system has portability and allows easy change or expansion since the whole system has been developed in FORTRAN.

ATHENA was inspired by a similar system developed for the Department of Energy by O'Neil and Sanders [5] called PERUSE. PERUSE was developed to aid in the analysis of large linear programming energy models. A study of the needs of Naval Postgraduate School students and faculty showed that additional capacities beyond those in PERUSE were necessary to support current and future research in large scale optimization. In addition to the standard MPS output, it was determined that ATHENA should support the experimental optimization system XS [3]; the output of this system contains in addition to standard MPS output, upper and lower penalties that implement the 'elastic formulation' of linear models that is unique to XS. ATHENA also had to support the



use of a preprocessor PREP [2] that reformulates the original optimization problem to an equivalent reduced problem with fewer rows and/or columns. A study of past and current modeling efforts at the Naval Postgraduate School identified additional commands that would help in the analysis of large models.

ATHENA was designed to be as much as possible a direct extension of PERUSE. Almost all the commands and options of PERUSE have been included with the identical names and syntax whenever possible. A summary of the extensions is listed under the section LIMITATIONS - EXTENSIONS, of the user's manual.

## II. SOFTWARE DESCRIPTION

### A. GENERAL

The whole system consists of 3 basic subsystems (see Figure 1).

#### 1. PARSER - Code Generator

This subsystem accepts as input a Query, parses it examining the syntax according to the productions of the Query Language and generates the corresponding internal code or gives information for syntax errors. The internal codes for each Query are shown in the program list.

#### 2. Input

This subsystem accepts as input either (1) an unpacked solution file in 'standard' format which it packs and saves for future reference, or (2) an L.P. solution file in packed form from a previous session.

#### 3. Interpreter

This subsystem, using the code generated from the PARSER, searches the packed solution file and prints out the information requested.

### B. LANGUAGE

#### 1. Host Language

The system has been developed in a portable subset of FORTRAN IV. FORTRAN was chosen for the following reasons:

a. FORTRAN is a general language available at almost any computer installation, so the system can be used with

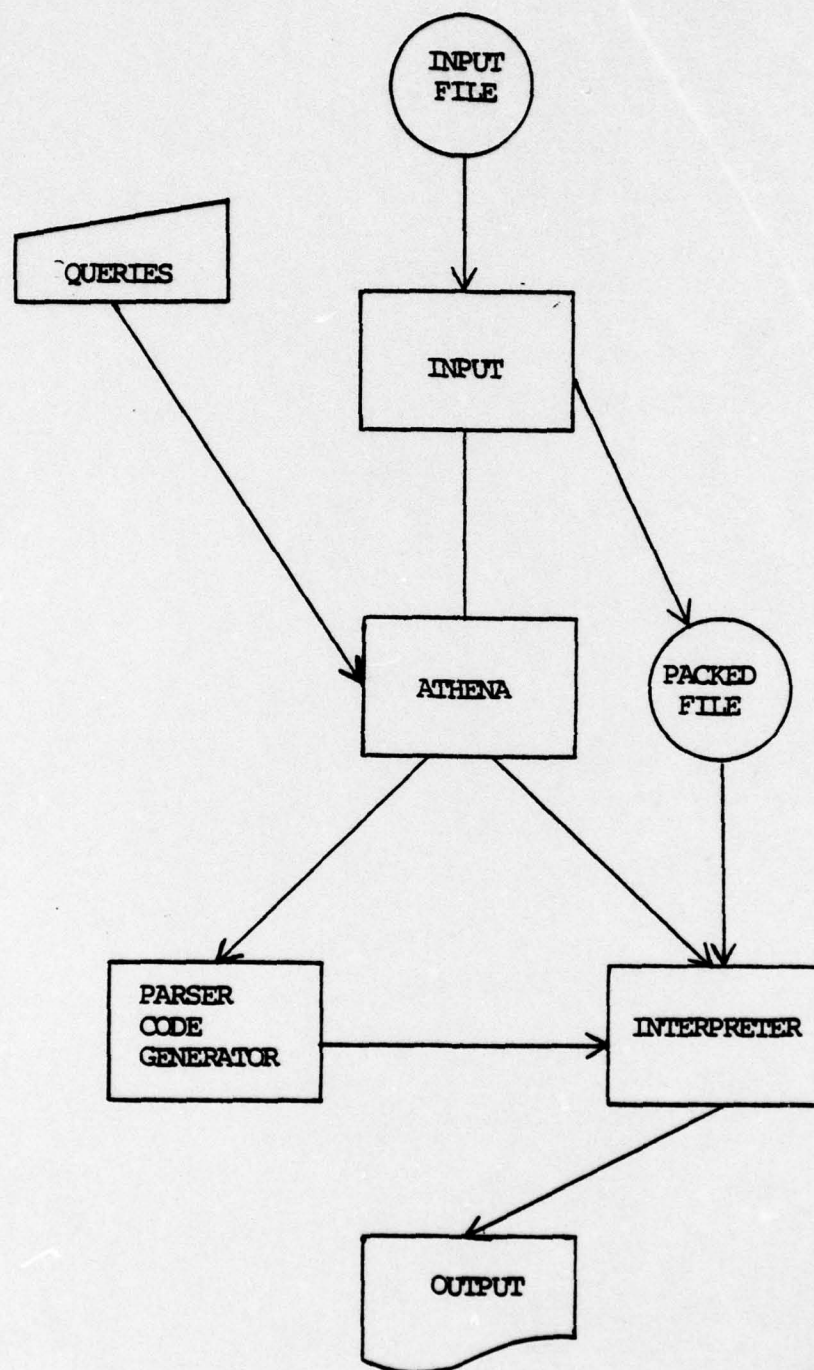


FIGURE 1. System ATHENA



any contemporary hardware. Non-IBM systems may require some program modifications.

b. Since FORTRAN is a high level language, the development time was low.

c. The response time for each Query is acceptable and there is little need for faster responses or enhanced efficiency.

d. Extensions and changes of the system can be easily implemented.

e. There is good system support for FORTRAN. In particular, ATHENA was developed with the FORTRAN G compiler of IBM 360/67.

## 2. Query Language

The set of acceptable Queries is divided into two main categories:

### a. Control Queries

Control Queries provide commands to the system to perform specific tasks, but usually do not use the solution file. Examples of Control Queries are those that accept comments for self documentation of the output, print headings for the output, terminate the use of system, etc.

### b. Command Queries

Command Queries use the solution file to extract the information asked for. Each Command Query consists of various fields separated by at least one blank. Some of the fields are optional, while others are required. An internal code number is generated by parsing the Command Query for

each field depending upon the analysis of that field and the previous fields in the Command Query.

The code generated by the PARSER is executed by the INTERPRETER which consists of a set of programs (subroutines) activated by the code numbers, to get the required information from the file.

## C. DATA STRUCTURES

### 1. Preview

There are some observations about the solution file of a linear program, especially of a large scale one, that lead to the use of a special data structure for storing a solution file in less memory space than it would otherwise require.

For each row or column the following information is usually included in the solution file.

NUMBER of row or column.

NAME, usually 6-8 alphanumeric characters.

STATUS, usually 2 characters, e.g., BS for BASIC, LL for LOWER LIMIT, etc.

ACTIVITY LEVEL, for each row or column.

SLACK ACTIVITY for rows or INPUT COST for columns.

LOWER LIMIT

UPPER LIMIT

DUAL ACTIVITY for rows or REDUCED COST for columns.

UPPER PENALTY and

LOWER PENALTY for the elastic linear programming system, XS [3].



There is redundant information in each record. Some of these redundancies are the following:

The explicit number for each row or column may be represented implicitly by the ordinal position of the row or column in the file. Rows usually precede columns in solution files.

A large number of the ACTIVITY LEVEL values will be zero. The same is true for the SLACK ACTIVITY, LOWER LIMIT, DUAL ACTIVITY and PENALTY values.

In many cases there will not be LOWER or UPPER LIMITS or PENALTIES.

PENALTIES in some cases may be infinite.

When the status of a row or column is 'fixed', then ACTIVITY LEVEL, LOWER and UPPER LIMITS are all the same number.

Each row or column can be in only one of its possible states.

Moreover, analysts who have experience with large scale Linear Programming have observed that most of the numbers of the solution file are the same. For example, most of the numbers for LIMITS are the same for a large number of rows or columns. For purposes of analysis, it is rarely necessary to have more than five decimal digits of precision for problem values. Indeed, some large problems cannot be solved with even this degree of significance. Accordingly, IBM single precision REAL\*4 representation

is adequate for our purposes. Conversion to REAL\*8 extended precision requires trivial program modifications.

Based on the above observations, two types of data structures for storing the solution file have been developed. The first one (SPARSE) exploits the redundant information into each individual record. The second (SUPERSPARSE) takes advantage of the last observation by storing each distinct number only once for the entire file. It is the responsibility of the user to select the data structure type that is appropriate for each solution file. SUPERSPARSE is probably superior for problems with less than half of all numbers possessing distinct real values. What follows is a detailed description of these two data structures.

## 2. SPARSE Data Structure

The entire solution file is stored in contiguous memory (8-bit bytes) as a one-dimension array called SOLFIL, in the following way:

a. The first 16 bytes (Four 4-byte words) are used to keep information for:

- (1) The size of the file in 4-byte words.
- (2) The type of data structure used to pack the file (SPARSE or SUPERSPARSE).

- (3) The number of rows and columns of the file.

b. For each row or column, 12 sequential bytes are required, organized as follows (see Figure 2).

- (1) The first 8 bytes hold the name of the row or column left justified, one character per byte.

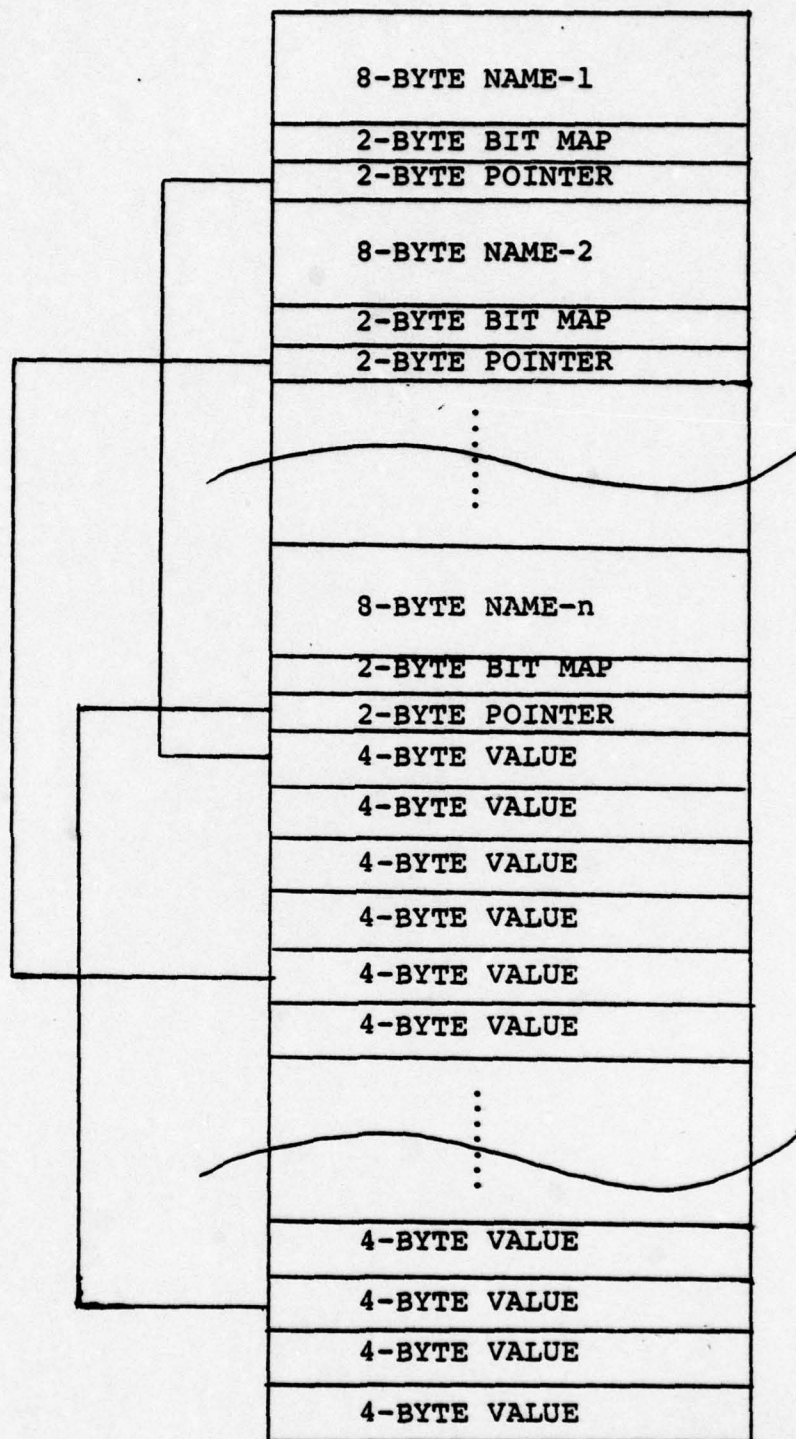


FIGURE 2. SPARSE Data Structure



(2) The next 2 bytes are used (as 16 bits) to represent various characteristics associated with that row or column.

(3) The last 2 bytes are used as a pointer to the first number stored from the current record.

c. The 16 bits from (2) above are organized in 4 groups of 4, 7, 4 and 1 bits, respectively, taken from higher to lower order.

The first group of 4 bits represents the status of the current row or column.

<u>BIT</u>	<u>PATTERN</u>	<u>STATUS</u>
0 0 0 0		IN (INFEASIBLE)
0 0 0 1		BS (BASIC)
0 0 1 0		LL (LOWER LIMIT)
0 0 1 1		UL (UPPER LIMIT)
0 1 0 0		EQ (FIXED)

The following status indicators are reserved for use with the program PREP [2]

0 1 0 1	VC (VOID COLUMN)
0 1 1 0	SC (SINGLETON COLUMN)
0 1 1 1	FC (FIX COLUMN)
1 0 0 0	BC (BOUND CHANGED)
1 0 0 1	VR (VOID ROW)
1 0 1 0	SR (SINGLETON ROW)
1 0 1 1	RR (REDUNDANT ROW)
1 1 0 0	FR (ROW FIXES VAR. AT BOUND)
1 1 0 1	ER (DOUBLETON EQUATION)

1 1 1 0

TR (TIGHTEN RANGE)

1 1 1 1

PP Reserved for PREP[2]

The next group of 7 bits represents the characteristic of zero or nonzero values for the record.

<u>BIT NO:</u>	<u>BIT VALUE:</u>	<u>CHARACTERISTIC</u>
11	0	ACTIVITY LEVEL NONZERO
	1	ACTIVITY LEVEL ZERO
10	0	SLACK/COST NONZERO
	1	SLACK/COST ZERO
0	0	LOWER LIMIT NONZERO
	1	LOWER LIMIT ZERO
8	0	UPPER LIMIT NONZERO
	1	UPPER LIMIT ZERO
7	0	DUAL/RED. COST NONZERO
	1	DUAL/RED. COST ZERO
6	0	LOWER LIMIT EXISTS
	1	LOWER LIMIT DOESN'T EXIST
5	0	UPPER LIMIT EXISTS
	1	UPPER LIMIT DOESN'T EXIST

The next group of 4 bits represents the characteristics for PENALTIES.

<u>BIT PATTERN</u>	<u>UPPER PENALTY</u>	<u>LOWER PENALTY</u>
0 0 0 0	ZERO	ZERO
0 0 0 1	ZERO	NUMBER
0 0 1 0	ZERO	INFINITY
0 0 1 1	NUMBER	ZERO

0 1 0 0	NUMBER	NUMBER
0 1 0 1	NUMBER	INFINITY
0 1 1 0	INFINITY	ZERO
0 1 1 1	INFINITY	NUMBER
1 0 0 0	INFINITY	INFINITY

The rest of the bit permutations are not used.

The last (0 bit) is used by the interpreter to mark the active and nonactive records when the user uses the ACTIVE or DEACTIVE commands to avoid searching of the entire file.

All the above groups of bits are stored together as a 16 bit binary number, which is stored in 2-byte halfword. ATHENA has provisions for the use of 16 bit halfwords representing absolute magnitudes of 0 - 65535, and can extract any component bits of the halfwords as necessary. (In this sense, the usual signed magnitude of IBM/360 halfword integers is ignored.)

d. The (nonzero, noninfinite) number values which must be stored are located immediately after all the information above. If the file represents a problem with M rows and N columns, then location INDEX - where  $INDEX = (N+M) * 3 + 4 + 1$  - of the SOLFIL array is the first eligible location for storing number values. The value of INDEX is kept in a 2-byte pointer associated with each row and indicates for that row the location of the first value stored. The sequence for storing these numbers for each row is:



ACTIVITY LEVEL, SLACK/INPUT COST, LOWER LIMIT, UPPER LIMIT, DUAL/REDUCED COST, UPPER PENALTY, LOWER PENALTY.

### 3. SUPERSPARSE Data Structure

This type of data structure takes advantage of the fact that in most problems many number values in the solution file are the same. Each distinct value is stored only once and a 2-byte pointer is used to access this value when needed. This is the only difference from the SPARSE representation (see Figure 3).

The array with the packed solution file is now separated into 3 parts:

- a. The first part is exactly the same as in SPARSE.
- b. The second part is substantially the same with the following differences:

- (1) It consists of 2-byte halfwords instead of 4-byte words.

- (2) Each halfword is a pointer to the third part of the array where the distinct number values are stored.

- c. The third part consists of a pool of 4-byte words, each representing a distinct real number value. The pointers to the distinct real number values are relative addresses in the real number pool, so a file which is packed with a different array size can be used with the current pointers providing the array size is large enough to hold the file.

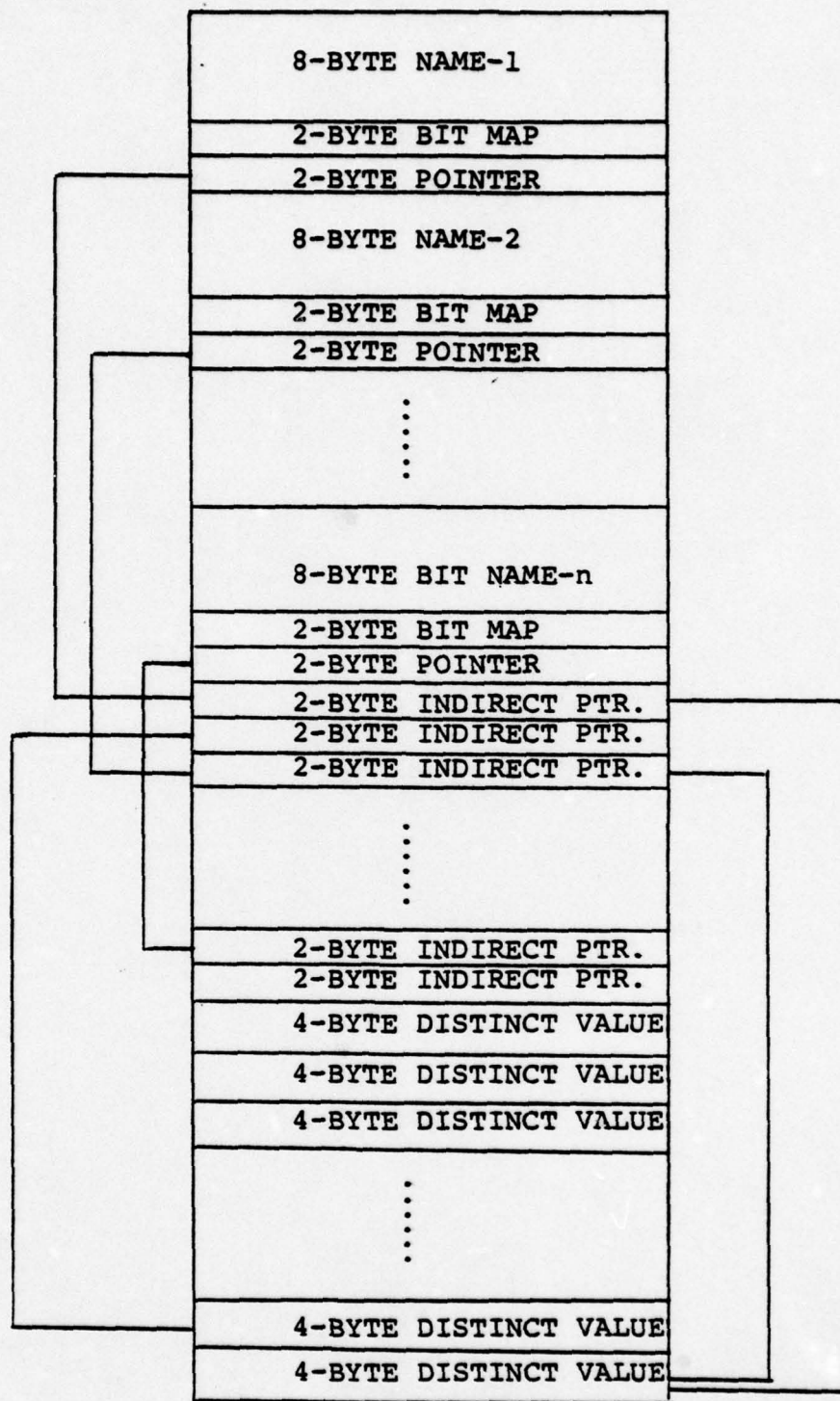


FIGURE 3. SUPERSPARSE Data Structure



### III. USER'S MANUAL

#### A. INTRODUCTION

The system ATHENA is a set of programs which accepts as input a linear programming solution file, packs it in a special data structure and interactively extracts specific information from that file through a set of Queries.

The size of memory which is required to run the system depends on the size of the file to be accommodated, and thus on the size of the original optimization problem. The user extracts information from the solution file with a Query Language, asking questions related to the solution of the problem represented in the file.

The entire system has been developed in FORTRAN language for portability and better coordination with other Linear Programming procedures which are also written in FORTRAN.

The Queries are self-documenting and their syntax follows closely the syntax of the English language. To avoid typing effort for experienced ATHENA users, short forms of Queries are provided. Only the characters comprising the short forms are interpreted by the system, with all subsequent contiguous nonblank characters ignored.

#### B. QUERY LANGUAGE

The Query Language consists of three subsets of Queries:

1. The Control Queries:

With this subset of Queries the user controls mainly the output of the system, inserting comments, headings, etc.

2. The Command Queries:

With these the user communicates with the solution file and extracts the specific information he needs.

3. The SET Command.

This command qualifies the ATHENA queries to access only a subset of the problem file.

1. Control Queries

- a. VERIFY (Short Form V)

All the following Queries will be displayed with the output. This Control Query is useful when the OFFLINE printer is used for the output instead of the terminal, or when the system is used under Batch Processing; in these cases answers are transmitted to the output device without the corresponding questions if the system is not in VERIFY mode.

- b. NOVERIFY (NOV)

The following Queries do not appear with the output. This Control Query is most frequently used when a terminal is used for all output. The DEFAULT mode of the ATHENA system is NOVERIFY.

- c. PROMPT (P)

The system responds with the prompt:

' INPUT A COMMAND '

whenever it is ready to accept a Query. PROMPT is a DEFAULT mode of the system.

d. NOPROMPT (NOP)

Used to avoid the prompting phrase in the output, especially when the OFFLINE printer or the Batch Processing is used.

e. H Any Character String

When the first column of a Query is the letter H, then the character string is printed as is in the output. H is used to insert comments or headings in the output.

f. \* Any Character String

When the first column of a Query is the character \*, no action takes place. This is considered as a comment and is ignored. \* is useful to insert comments and/or headings on the terminal output.

g. END (E)

Used to end the current session.

2. Command Queries

A Command Query consists of several fields. Some fields are required and must always appear in a Command Query and others are optional. Each field is separated from the others by at least one blank character. The number of blanks between fields is not significant and a Query may start at any character position in the command. The length of a Query cannot exceed 80 characters including the spaces between the fields.



The possible fields that can be included in a Command Query are:

- a. TYPE
- b. SELECT
- c. MASK
- d. CONDITION
- e. PRINT OPTION

The fields in a Command Query must appear in the above sequence and the first 3 of them must always appear, with only 2 exceptions. A detailed description follows of each individual field and the way that it may be used.

a. TYPE field

This is the first field of the Query and may start at any character position. This field can be one of the following:

(1) DISPLAY (D)

Used when all the records which meet the requirements of the other fields are to be displayed in the output in the sequence they are encountered starting from the beginning of the solution file.

What portion of each individual record will be displayed depends on the PRINT OPTION field.

(2) COUNT (C)

Used when only the number of records which meet the requirements of the other fields is desired. COUNT is especially useful immediately preceding a DISPLAY command so the user will know in advance the size of output, avoiding

unpredictably extensive printouts. For this Query the PRINT OPTION is ignored as meaningless.

(3) ADD (A)

Used when some numerical quantities of the qualified records are to be summed. The names of the numeric quantities of each record that will be added are given in the PRINT OPTION field. If no PRINT OPTION appears, all the numeric quantities of each record are added and their sums are displayed with appropriate labels.

Since it is mathematically meaningless to add LOWER or UPPER LIMITS, or PENALTIES, they can not be summed or displayed.

(4) AVERAGE (AV)

Used exactly as the ADD command to display arithmetic averages. The sums are divided by the total number of the qualified records.

(5) ACTIVATE (AC) (Syn. ACTIVE)

With the ACTIVATE command the user can indicate a subset of the records of the solution file with specific qualifications determined by the other fields so that subsequent Queries will implicitly refer only to that subset. The user can expand the initial subset by using the ACTIVE command repeatedly to add new records to the active subset.

The command ACTIVATE can minimize the searching time for the required information in the active subset. Each time the ACTIVATE command is issued, the system

responds with the number of records added to the active subset and the current total number of active records.

(6) DEACTIVATE (DE) (Syn. DEACTIVE)

Used to delete records with specific qualifications from the current active subset - created by the ACTIVE commands - or to eliminate any active file. The system responds with the number of records deactivated and the total number of records remaining active.

The entire active file can be deactivated by:

' DEACTIVE ALL or DE A'.

With this Query all the currently active records will be deactivated and the message:

' F I L E D E A C T I V E '

will be printed out. Subsequent queries will refer to the entire solution file.

b. SELECT field

This field is mandatory and specifies whether the qualified records are ROWS, COLUMNS or BOTH. It may consist of one of the following:

(1) ALL (A)

Specifies that the entire file must be searched for the qualified records starting from the first ROW and continuing to the last COLUMN.



(2) COLUMNS (C)

Specifies that the COLUMNS only will be searched for the qualified records starting with the first COLUMN and continuing to the last COLUMN.

(3) ROWS (R)

Specifies that the ROWS only will be searched for the qualified records starting with the first ROW and continuing to the last ROW.

c. MASK field

Specifies that any record is qualified for processing if the name of the record fits the MASK field. The MASK field is left justified and may contain 1 to 8 characters. All the right unfilled positions up to 8 characters are assumed to be the character \*. The MASK is matched against the name, starting from the left, character by character. Any character in the name is matched with a \* in the MASK field. The MASK field is mandatory.

EXAMPLES

i. The MASK 'X\*\*\*\*\*Y' specifies all the names starting with the letter X and having as the 8th (last) character the letter Y.

ii. The MASK 'X' is equivalent with the MASK 'X\*\*\*\*\*' and means all the names starting with the letter X.

iii. The MASK '\*\*\*\*\*Y' specifies all the names ending with the letter Y and it is NOT equivalent with the MASK 'Y'.

iv. The MASK 'ABCDEFXY' specifies only this name and since the names of ROWS and COLUMNS are assumed to be inclusively unique, the searching of the file stops when the first match is made.

v. The MASK '\*' specifies ALL the names and may be used when no particular mask is desired.

d. CONDITION field

The syntax of this field is:

FOR ( <conditional phrase> )

The word FOR, left parenthesis and right parenthesis must always appear when the CONDITION field appears in a Query.

There are two kinds of conditional phrases:

The simple conditional phrase and the compound conditional phrase.

(1) Simple Conditional Phrase

There are 3 kinds of simple conditional phrases: The Relational, the Status and the Bound simple conditional phrases.

(a) Relational Simple Conditional Phrase

The syntax is:

<Arg1> <Relop> <Arg2>

where Arg1, Arg2 and Relop are one of the following:

i. Arg1

X	for	ACTIVITY LEVEL
S or C	for	SLACK ACTIVITY or INPUT COST



L	for	LOWER LIMIT
U	for	UPPER LIMIT
D	for	DUAL ACTIVITY or REDUCED COST
P	for	UPPER PENALTY
W	for	LOWER PENALTY

#### ii. Relop

Relational operators EQ, NE, GT, GE, LT, LE with the same meaning as in FORTRAN. (Note, however, that there are not imbedding decimal characters as in FORTRAN.)

#### iii. Arg2

Arg2 is defined exactly as Arg1 with the enhancement that Arg2 may also be any integer or real number. Arg2 cannot be expressed as a floating point number in exponential notation.

#### (b) Status Simple Conditional Phrase

The syntax is:

STATUS <Flag> or ST <Flag>

where Flag is one of the following:

BS	for	BASIC
LL	for	LOWER LIMIT
UL	for	UPPER LIMIT
EQ	for	FIXED
VC	for	VOID COLUMN
SC	for	SINGLETON COLUMN

FC	for	FIXED COLUMN
BC	for	BOUND CHANGED
VR	for	VOID ROW
SR	for	SINGLETON ROW
RR	for	REDUNDANT ROW
FR	for	FREE ROW
ER	for	DOUBLETON EQUATION
TR	for	TIGHTEN RANGE

(c) Bound Simple Conditional Phrase

The syntax is:

<Arg1> MINIMUM or <Arg1> MAXIMUM

where Arg1 is specified as in Relational Simple Conditional Phrase. The words MAXIMUM or MINIMUM can be abbreviated as MAX or MIN, respectively. This is used to extract those records which have the MAXIMUM or MINIMUM value in the specified field with the specified MASK. The system responds with the first record encountered with the maximum or minimum value associated with Arg1, and the total number of records that meet the requirements. This phrase may not be used with ACTIVE or DEACTIVE options in the TYPE field of the Query.

(2) Compound Conditional Phrase

The syntax of this phrase is:

<Relational Cond. Phrase> <Log. Oper.> <Relational Cond. Phrase>

or

<Relational Cond. Phrase    Log. Oper.    Status Cond. Phrase>

where Log. Oper. is OR or AND with the meaning of the corresponding logical operators. Note that the Bound Simple Conditional Phrase is not compatible for use in a Compound Conditional Phrase, since it exhibits no boolean value. Also, the Status Conditional Phrase must always appear after the logical operator.

The CONDITION field as a field must be separated by at least one blank from the other fields of the Query. The word FOR, the left parenthesis, and the first element of the conditional phrase do not require separation by blank characters, nor do the last element of the conditional phrase and the right parenthesis.

The CONDITION field is optional and need not appear in the Query. If it is not present, any record is qualified if the MASK field is satisfied. Using the ACTIVATE and DEACTIVATE commands the user can actually have unlimited length conditional phrases, by adding qualified subsets of records in the ACTIVE file.

e. PRINT OPTION field

This is the last field of a Query. It is optional, and if it does not appear the entire record which satisfy both the MASK and the CONDITION fields are printed out.



The elements of the PRINT OPTION field may be any combination of the following:

X, C or S, L, U, D, P, W

with meanings as described in the CONDITION field. The output will include information described in the PRINT OPTION with corresponding headings. The elements of the field can be separated by any number of blanks, by commas, or not at all. If both C and S appear in the PRINT OPTION neither of them is printed out. For the commands ADD and AVERAGE the default PRINT OPTION is X, C or S, D since there is no meaning for LIMITS and PENALTIES.

For all Queries that potentially require more than one output record for the answer (i.e., all Queries except COUNT, ACTIVE, DEACTIVE and SET), the output will include the following entries for each record: NUMBER, NAME, STATUS and the entries specified in the PRINT OPTION field in the sequence in which they appear. At the end of the answer output for each Query the total number of qualified records is given. The heading for the output is determined by the SELECT field. If for this field the option ALL is used, the heading will be the one for ROWS although COLUMNS may also be included in the output.

### 3. The SET Command

By default each time a Command Query is issued the whole solution file is searched starting at the first ROW or

COLUMN and continuing by examining sequentially all the records.

Queries may sometimes apply only to a small part of the solution file or to records whose relative position in the file is known. In these cases the SET command can cause searching to be initiated at a particular entry in the file and continued to another particular entry. Also a fixed step size can be specified for the search. Thus much computational effort can be avoided.

The syntax for the SET Command is:

SET <number 1> <number 2> <number 3>

where;

number 1 is the number of the starting record

number 2 is the number of the record to stop  
searching

number 3 is the step for searching.

All these numbers must be integers separated by at least one blank and the presence of all of them is required. These numbers also must be in the range of total number of records for the file. The SET limits apply to qualify any subsequent search of the file even if ROW or COLUMN subsets are specified by a Query.

#### EXAMPLE

Suppose the solution file has 300 rows and 2000 columns and the following SET command is issued:

SET 18 1500 10

For all subsequent Queries:

If the SELECT field of the Query is ALL then the searching starts at the 18th record and continues through the 1500th record with step 10 (i.e., Record numbers 18, 28, 38, .... are examined).

If the SELECT field is ROWS then the searching will start at the 18th row through the last row (300th) with step 10.

If the SELECT field is COLUMNS then the searching will start at 18th column through the 1500th column (or equivalently the 318th record through the 1800th record) since the number of columns is greater than 1500.

To restore default settings, use:

'SET DEFAULT' or 'SET D'

#### C. ERROR MESSAGES

The following error messages are typed at the terminal as soon as they are detected. If the error is only in syntax, the system is immediately ready to accept a new query, otherwise execution is terminated. Errors have been grouped with one message for each group. Messages are self explanatory.

<u>ERROR NO</u>	<u>POSSIBLE REASON</u>
1	: Attempt to parse a blank query.
101	: Invalid TYPE field. One of the characters D,V,C or blank was expected after A.



102 : Invalid TYPE field. One of the characters  
O,E was expected after S.

103 : Invalid TYPE field. One of the characters  
P,V was expected after NO.

104 : Invalid TYPE field. No command starts  
with the given letter.

201 : Missing character or somewhere in the  
query there is no space delimiter.

202 : There is no space delimiter.

301 : Invalid SELECT field. SELECT field is  
missing or there is no space delimiter  
between TYPE and SELECT fields.

502 : Invalid CONDITION field. the word FOR is  
missing (the string OR was expected after  
F), or invalid PRINT field.

503 : Missing left parenthesis in CONDITION field.

504 : Incomplete condition field or missing  
space delimiter.

505 : Missing right parenthesis in condition field.

506 : Invalid OR logical operator. Character R  
R was expected after O.

507 : Invalid AND logical operator. The string  
ND was expected after A.

508 : Invalid logical operator. Only OR and  
AND are accepted.

511 : Invalid operand for status. The character  
C or S was expected after B.

512 : Invalid operand for status. The character  
L was expected after L.

513 : Invalid operand for status. The character  
L was expected after U.

514 : Invalid operand for status. The character  
Q or R was expected after E.

515 : Invalid operand for status. The character  
V, S, F or B was expected before C.

516 : Invalid operand for status. The character  
V, S, R, F, E or T was expected before R.

517 : Invalid operand for status. The character  
P was expected before P.

518 : Invalid operand for status. The character  
I, A, C or D was expected before E.

519 : Missing space delimiter after status  
operand.

520 : Non recognizable operand for status.

601 : Invalid first operand for relational  
operator in condition field.

602 : Missing space delimiter in a simple  
conditional phrase.

603 : Invalid relational operator. The character  
T or E was expected after G.

604 : Invalid relational operator. The character  
T or E was expected after L.

605 : Invalid relational operator. The character  
Q was expected after E.

606 : Invalid relational operator. The character  
E was expected after N.

607 : Invalid operand in bound conditional  
phrase. The character N was expected  
after string MI.

608 : Invalid operand in bound conditional  
phrase. The string AX was expected after M.

609 : Unrecognizable relational operator in  
simple conditional phrase.

701 : Invalid print field, or missing word FOR  
in condition field.

1001 : Error in input data. Unrecognizable  
status code.

1002 : Error in input data. Data encountered  
has less than the expected number of rows  
and columns.

#### D. LIMITATIONS - EXTENSIONS

As mentioned in the introduction, ATHENA is a direct expansion of the PERUSE system. It includes all the features of PERUSE, except the weighted average command and has the following differences and extensions:

1. ATHENA supports two distinct data structures, each different from that of PERUSE. This was necessary in order to support efficient access to individual records or group of records. The SUPERSPARSE data structure is unique to ATHENA.



2. ATHENA accepts as input a simple file which can be easily obtained from the solution file of any linear programming package on tape, disk or cards.

3. ATHENA supports the commands SET, ACTIVATE, DEACTIVATE and COUNT, in addition to the commands of PERUSE, allowing the user to construct logical subsets of the solution and efficiently access these subsets as independent files with very small access time.

4. ATHENA supports compound conditional phrases for extraction of more specific information and the bound conditional phrase for maximum and minimum values.

5. ATHENA uses object time variable format allowing better appearance of output and uses the words NONE and INFINITY instead of the number 0.7273E76 for better readability.

6. ATHENA accepts reduced problems from PREP [2] and can be used to pass the PREP status file with the solution file of any optimization system to permit recovery of the original problem solution.

ATHENA has been designed to handle solution files with up to 30,000 records. The actual limit is imposed by the number of real number values that must be stored explicitly. This number cannot presently exceed 65536 since this is the largest integer pointer value which can be stored by ATHENA in a 2-byte halfword. Experience has shown that the average number of stored values for each record, excluding penalties

is 1.5 [7]. Adding to this another 0.5 per record for penalties, the problem size limit may be as large as 30,000 rows and columns.

A rough estimation of the space needed for the packed file in 4-byte words can be obtained by multiplying the sum of rows and columns of the solution file by 5 for the SPARSE data structure and by 4 for SUPERSPARSE. Before using ATHENA, adjust the size of the SOLFIL array in common block SOLPAC to this number. To avoid passing problems with common areas under the IBM 360/67 Operating System, use an array size that is an exact multiple of 4096 larger than the number calculated above. Also make corresponding adjustments to the DEFINE FILE statement of the main program. Other computers will require analogous modifications to this array size.

ATHENA has been developed in modular form and can be easily changed or extended to support future needs. Commands which can be easily implemented include the weighted average, the sort of output, or further calculations needed for the analysis of the solution file. ATHENA can also be used as part of an integrated system for sensitivity analysis of optimization problems.

## E. SYSTEM INTERFACE

### 1. Preview

Linear Programming packages give differing forms of output so that it is difficult for a system to be interfaced

with all of these solution formats. ATHENA accepts as input a solution file in a 'standard' form which can be easily obtained from any other solution file form.

ATHENA is best utilized in an interactive system such as CP/CMS, although it can also be used in batch processing. On the other hand, most L.P. packages run only in batch processing. Moreover, in some systems there is no integration of interactive and batch processing (as is currently the case at the Naval Postgraduate School Computer Center). In these cases, the solution files may be transferred manually from one system to the other using magnetic tapes or cards.

A simple input file has been designed which can be punched in cards or entered on tape, disk, or other storage media.

## 2. Input File

The input file consists of records with the following structure:

a. The first record always contains the number of ROWS, the number of COLUMNS, and in position 51 the character '1' if each ROW record contains PENALTIES or '0' otherwise. The FORMAT of the first record is

(I5,30X,I5,10X,I1).

b. Each subsequent record contains explicitly all the information associated with each ROW and COLUMN, with



the following format:

NAME        Format    2A4    (left justified)

STATUS      Format      A2

X, C or S, L, U, and D numeric field values with Format 5E14.5 or 5F14.5. (The meanings of each of these fields is described in the previous section.) If the solution file includes PENALTIES, then two records will be associated with each ROW. The first will be exactly that described above, and the second will have the FORMAT(E14.5,16X,E14.5) for P, and W, the UPPER and LOWER PENALTIES. In all cases, INFINITE values will be represented explicitly by the number  $\pm 0.1E76$ . The total number of records must agree with the sum of ROWS plus COLUMNS, with the records of ROWS preceeding those of COLUMNS; otherwise an Input error will occur. The file is read in and packed one record at a time.

### 3. Packed File

#### a. Packed File as Input

If the input file is already packed from a previous use of the system, it will be read in unformatted binary format. The system will provide the user information for memory requirements before reading the file. The packed file may be on a tape or disk but cannot be on cards. The system will ask the user at the beginning of a session for the number of the file.

#### b. Packed File as Output

If an unpacked file is used as input, the system will ask for the file number where a packed file is to be

written. Of course, a DEFINE FILE FORTRAN statement must be included right after the declarations of the main program.

#### E. EXAMPLE OF SYSTEM USE

The procedure follows for use of ATHENA at the Naval Postgraduate School Computer Center with the IBM 360/67. The solution file here is produced by the MPS/360 package and ATHENA is used under CP/CMS. Similar procedures can be followed for any other installation.

##### 1. Obtaining the Unpacked Solution

a. Submit the problem to be solved using the usual Control Cards required for the MPS/360 package inserting before the Control Card:

```
//MPS2.SYSIN DD *
```

the following cards:

```
//MPS2.SYSPRINT DD DSN=Sxxxxx.nnnnnn,  
// UNIT=3330,VOL=SER=DISK04,  
// SPACE=(CYL,(1,1)),DISP=(NEW,KEEP),  
// DCB=(RECFM=UA,BLKSIZE=133)
```

With these cards the output of MPS will go to the DISK instead of the printer.

xxxx is the user's number and  
nnnnnn is the file name on the disk.

If the solution file is too big or disk space is not available use tape or tapes to store the output.

b. Use the program REWRITE (page 98) to transform the MPS/360 tape or cards to the format required for the ATHENA unpacked Input file.

c. Now the unpacked file for ATHENA is available and can be used to analyze the solution.

## 2. Using the System Under CP/CMS

ATHENA in CP/CMS TEXT form requires about 67K bytes. The space needed for the packed file depends on the number of records, the method used for packing and the density of the original file. 200K bytes would be sufficient to hold a packed file with up to 13,000 rows and columns. After sufficient space has been secured, the following procedure may be applied:

a. Ask OPERATOR to connect the tape with the Input file created by the REWRITE to the private disk as device 181. As soon as the tape is connected, the message 'DEVICE 181 ATTACHED' will be printed at the terminal.

b. Before using the tape, type ALWAYS under CMS the command 'TAPE SKIP 1'. This command will position the tape at the first record of the file. This command is required because the tape created by IBM O.S./360.

c. Type \$ ATHENA

ATHENA will ask for information about the file identifiers for input-output, whether the file is packed and method of packing and will give the size of the packed



file. For input file ordinal use any number between 01 and 99 excluding the numbers 03 - 06. For output file ordinal give the number 03 or 04. These are the numbers used by the DEFINE FILE FORTRAN statement and they can be changed. The system will be ready to accept QUERIES as soon as the prompt phrase 'INPUT A COMMAND' is typed by ATHENA at the terminal. The packed file can be saved on a tape using the 'TAPE DUMP' command under CMS.

In the next few pages a demonstration of using ATHENA with a solution file of 766 rows and 10921 columns is given. This problem is a mixed integer optimization model with 963 binary variables for medium term capital budgeting of the Naval Air Test Center [6]. For this problem, a query may require as much as one and a half minutes of clock time if the interactive system on the IBM 360/67 is under heavy use and the query is difficult to answer. However, most queries are answered almost immediately. Response time is especially good when the user makes use of ACTIVATE, MASK and SET features to qualify necessary searching.

```

$ ATHENA
  WHAT IS THE FILE NO OF THE SOLUTION FILE ? (FORMAT I2)
04
  IS THE FILE ALREADY PACKED ? ENTER YES OR NO :
YES
    MEMORY REQUIRMENTS  FOR THE  PACKED FILE  :47749  4-BYTE
WORDS

  IF YOU HAVE SUFFICIENT MEMORY SPACE ENTER YES

  OTHERWISE ENTER  NO  MAKE  ADJUSTMENTS AT  COMMON AND  TRY
AGAIN
YES
      INPUT A COMMAND
*****
*   USING THE CHARACTER * AS THE FIRST COLUMN OF THE QUERY *
*   THE QUERY IS IGNORED AND THIS IS A CONVENIENT WAY TO   *
*   INSERT COMMENTS IN THE OUTPUT FROM A TERMINAL.         *
*   COMMENTS IN THE OUTPUT FROM THE OFFLINE PRINTER ARE    *
*   INSERTED USING THE LETTER H INSTEAD OF *.              *
*   TO AVOID THE PROMPTNESS PHRASE 'INPUT A COMMAND' AT     *
*   THE OUTPUT USE THE COMMAND 'NOPROMPT'.                 *
*   BECAUSE OF THE REQUIREMENTS FOR THE APPEARANCE OF THE  *
*   THESIS THE PRINT OPTION FIELD OF THE DISPLAY COMMAND    *
*   IS USED ONLY WITH AT MOST TWO OPTIONS.                 *
*****
*
*
*   HERE IS A DEMONSTRATION OF USING THE SYSTEM ATHENA.
*
*
*   HOW MANY ROWS HAS THE FILE ?
*
COUNT ROWS *

      766  ROWS OR COLUMNS WITH MASK : *****
SATISFY THE CONDITIONS
*
*   HOW MANY COLUMNS ?
*
COUNT COLUMNS *

      10921  ROWS OR COLUMNS WITH MASK : *****
SATISFY THE CONDITIONS
*
*   HOW MANY OF THEM ARE BASIC ?
*
      C   C   *   FOR(ST BS)

```

506 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*  
\* HOW MANY OF THEM ARE EQUAL ZERO ?  
\*  
C C \* FOR( X EQ 0. AND ST BS)

55 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*  
\* USING THE COMMAND 'ACTIVATE' ONLY THE ACTIVATED  
\* RECORDS ARE SEARCHED TO ANSWER THE QUESTION.  
\* THIS IS A GOOD WAY TO AVOID SEARCH OF THE WHOLE  
\* FILE.

\*  
ACTIVATE COLUMNS X0

73 RECORDS ACTIVATED  
TOTAL RECORDS ACTIVE : 73

\*  
\* HOW MANY ROWS NOW ?  
\*  
C R \*

0 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*  
\* THERE ARE NO ROWS SINCE ONLY COLUMNS ACTIVATED.  
\* HOW MANY OF THEM ARE BASIC ?  
\*

C C \* FOR(ST BS)

1 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*  
\* DISPLAY THEM  
\*

DISPLAY C \* FOR(ST BS) X L  
THE FOLLOWING ROWS OR COLUMNS SATISFY CONDITIONS

NUMBER	.NAME..	AT	...ACTIVITY...	..LOWER LIMIT.
803	X041	BS	0.12910	0.00000

1 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*  
\* WHAT IS THE AVERAGE OF NONZERO ACTIVITIES ?  
\*  
AVERAGE C \* FOR(X NE 0)  
THE FOLLOWING ROWS OR COLUMNS SATISFY CONDITIONS



NUMBER	.NAME..	AT	...ACTIVITY...	..INPUT COST..
SUMS OR AVERAGES :			0.98773	0.76576

71 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
 SATISFY THE CONDITIONS

\*  
 \* THE SUM OF ACTIVITIES AND INPUT COST ?  
 \*

ADD C \* FOR ( X NE 0 )  
 THE FOLLOWING ROWS OR COLUMNS SATISFY CONDITIONS

NUMBER	.NAME..	AT	...ACTIVITY...	..INPUT COST..
SUMS OR AVERAGES :			70.12909	54.36882

71 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*

SATISFY THE CONDITIONS

\*  
 \* ADD SOME NEW COLUMNS AT THE ACTIVE FILE  
 \*

ACT C YM01

175 RECORDS ACTIVATED  
 TOTAL RECORDS ACTIVE : 248

\*  
 \* BASIC ?  
 \*  
 C C \* FOR ( ST BS )

7 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
 SATISFY THE CONDITIONS

\*  
 \* DISPLAY BASIC AND ZERO ACTIVITY  
 \*

D C \* FOR ( X EQ 0 AND STATUS BS ) X C  
 THE FOLLOWING ROWS OR COLUMNS SATISFY CONDITIONS

NUMBER	.NAME..	AT	...ACTIVITY...	..INPUT COST..
5568	YM01061	BS	0.00000	-0.01000

1 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
 SATISFY THE CONDITIONS

\*  
 \* HOW MANY INPUT COSTS ARE ZERO ?  
 \*

C C \* FOR ( C EQ 0 )

5 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*

\* HOW MANY AT LOWER LIMIT ?

\*

C C \* FOR (STATUS LL)

171 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*

\* ELIMINATE SOME RECORDS FROM THE ACTIVE FILE

\*

DEACTIVATE C X

73 RECORDS DEACTIVATED  
TOTAL RECORDS ACTIVE : 175

\*

\* CHECK FOR THE ELIMINATION

\*

C C X

0 ROWS OR COLUMNS WITH MASK : X\*\*\*\*\*  
SATISFY THE CONDITIONS

\*

\* ELIMINATION O.K.

\*

\* HOW MANY AT LOWER LIMIT (LL) NOW ?

\*

C C \* FOR (ST LL)

169 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*

\* WHAT IS THERE AVERAGE ?

\*

AV C \* FOR (ST LL)

THE FOLLOWING ROWS OR COLUMNS SATISFY CONDITIONS

NUMBER	.NAME..	AT	...ACTIVITY...	..INPUT COST..
SUMS OR AVERAGES :			0.00000	-0.01000

169 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*

\* RETURN TO THE ORIGINAL BIG FILE

\*

DEACTIVATE ALL

F I L E D E A C T I V E

\*  
 \* DISPLAY THE 5 FIRST ROWS  
 \*

SET 1 5 1  
 D R \* X S  
 THE FOLLOWING ROWS OR COLUMNS SATISFY CONDITIONS

NUMBER	.NAME..	AT	...ACTIVITY...	SLACK ACTIVITY
1	BENEFITS	BS	449.49902	-449.49902
2	RC1011	UL	0.00000	0.00000
3	RC1021	UL	0.00000	0.00000
4	RC1031	UL	0.00000	0.00000
5	RC1041	BS	0.00000	0.00000

5 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
 SATISFY THE CONDITIONS

\*  
 \* THE FIRST 5 COLUMNS  
 \*

D C \* C D  
 THE FOLLOWING ROWS OR COLUMNS SATISFY CONDITIONS

NUMBER	.NAME..	AT	..INPUT COST..	.REDUCED COST.
767	X001	UL	0.99994	0.36312
768	X002	UL	0.89108	0.77450
769	X003	UL	0.45629	0.23338
770	X004	UL	0.45483	0.36919
771	X005	UL	0.73440	0.60813

5 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
 SATISFY THE CONDITIONS

\*  
 \* RETURN TO THE ORIGINAL FILE  
 \*

SET DEFAULT

\*  
 \* DISPLAY VARIABLES IN A SPECIFIC RANGE  
 \*

D C YM FOR (X GT 10.5 AND X LT 25.5) X L  
 THE FOLLOWING ROWS OR COLUMNS SATISFY CONDITIONS

NUMBER	.NAME..	AT	...ACTIVITY...	..LOWER LIMIT.
5708	YM05061	BS	14.00000	0.00000
6139	YM17171	BS	11.54310	0.00000
6247	YM20201	BS	17.13029	0.00000
6758	YM35061	BS	17.29999	0.00000
7472	YM20202	BS	18.98279	0.00000
7983	YM35062	BS	11.90000	0.00000
8697	YM20203	BS	21.00000	0.00000
9208	YM35063	BS	16.59999	0.00000
9378	YM05014	BS	11.90000	0.00000
9922	YM20204	BS	13.00000	0.00000
10433	YM35064	BS	23.06062	0.00000
10608	YM05065	BS	14.00000	0.00000
11147	YM20205	BS	13.00000	0.00000
11658	YM35065	BS	19.50000	0.00000



14 ROWS OR COLUMNS WITH MASK : YM\*\*\*\*\*  
SATISFY THE CONDITIONS

\*

\* WHAT IS THE MAXIMUM OF THE ACTIVITY LEVEL ?

\*

D C \* FOR(X MAX) X C

1 YM18184 BS 88.63869 0.00000

1 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*

\* THE MINIMUM LOWER LIMIT FOR THE ROWS ?

\*

D R \* FOR(L MIN) L U

766 BENEFITS BS NONE NONE

766 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*  
SATISFY THE CONDITIONS

\*

\* THE MAXIMUM ACTIVITY LEVEL FROM VARIABLES

\* STARTING WITH LETTER Y ?

\*

D C Y FOR ( X MAX ) XC

1 YM18184 BS 88.63869 0.00000

1 ROWS OR COLUMNS WITH MASK : Y\*\*\*\*\*  
SATISFY THE CONDITIONS

\*

\* ARE THERE INFEASIBILITIES ?

\*

COUNT ALL \* FOR (ST IN)

0 ROWS OR COLUMNS WITH MASK : \*\*\*\*\*

SATISFY THE CONDITIONS

\*

\* EXIT FROM THE SYSTEM

\*

END

## APPENDIX

The whole program consists of 3 main parts: The input part, the parser-code generation part and the interpreter part (see Figure 1). The main program calls these 3 parts according to the instructions from the user. More detailed information for the main subroutines of each part are given below.

### 1. Input Part.

There are 2 almost identical subroutines named READF and READD. READF is used to pack the solution file in SPARSE data structure and READD in SUPERSPARSE.

#### a. Input Parameters

(1) IFLR: The file number of the solution file. This can be any number between 1 - 99 excluding the numbers 3, 4, 5, and 6.

(2) IFLW: The file number where the packed file will be written. Currently this number can be 3 or 4 since the DEFINE FILE statement in the main program defines the files 3 and 4 only.

#### b. Output Parameters

(1) NROWS, NCOL; The number of rows and columns of the solution file respectively.

(2) IER: If this value is different from zero an error has been encountered in the input data.

The solution file is read in standard format, as explained in user's manual, one record at a time, filling the array SOLFIL in the common area SOLPAC with the packed file. The name of the record is read directly into SOLFIL and the associated values into the array DATA. The 2 character status code is read into the INTEGER\*2 variable STAT. The INTEGER\*2 array CSTAT has all possible values for status, and their location in this array, minus one, gives the corresponding value of the first bit group in the 16 bit map as explained in the data structure section.

The variable INDEX always points to the next available location for storing the next nonzero, noninfinite number. The value of INDEX at the beginning of the processing of each record is stored in the 2-byte pointer after the name. Each number in the array DATA is examined and if it is eligible for storage it is stored at the location SOLFIL(INDEX) and INDEX is increased by one. At this point, for the subroutine READD, the end of the array SOLFIL, where the distinct real number values are stored, is searched and if the value sought is found the location of this value, counting from the end of the array is stored at the location SOLFIL(INDEX); otherwise, a new entry is made and the location of this entry is stored at the location SOLFIL(INDEX). Notice that SOLFIL(INDEX) is a 4-byte location for READF but 2-byte for READD as explained earlier. After the last record has been packed, READD shifts the distinct real number values



down to the end of 2-byte indirect pointers. When the system is to be used under batch processing the CALL TAPSET statement from READF and READD must be removed and the input file must be described using the usual DD statement.

## 2. Parser - Code Generator

There are several subroutines in this part of the system. There is one subroutine for each field of the query, the control subroutine PARSE and various others to perform specific tasks like getting the numerical value of a string of decimal characters, etc. The subroutine PARSE is described below. The numerical codes used for each case have been inserted as comments at the beginning of each subroutine where they are used.

### a. Input Parameters

(1) QRY: INTEGER\*2 array with the query to be parsed in A format.

### b. Output Parameters

(1) CODE: INTEGER\*2 array with the numerical code for query as follows:

QRY(1)	: TYPE	field
QRY(2)	: SELECT	field
QRY(3)	: MASK	field
QRY(4) - QRY(10)	: CONDITION	field
QRY(11) - QRY(18)	: PRINT	field

(2) MASK: INTEGER\*2 array with the mask from the query, one character per location.

(3) ARG2, ARG4 : numerical values from the condition field, if any.

(4) IER : Code number for syntax error. If IER=0 the query has been parsed correctly.

Subroutine PARSE calls sequentially the corresponding subroutines to parse each field of the query. Before calling it positions the pointer ICOLL at the location of the QRY where the field starts using the subroutine GETNCH.

### 3. Interpreter Part

There are several subroutines called by the main subroutine INTERP of this part.

#### a. Input Parameters

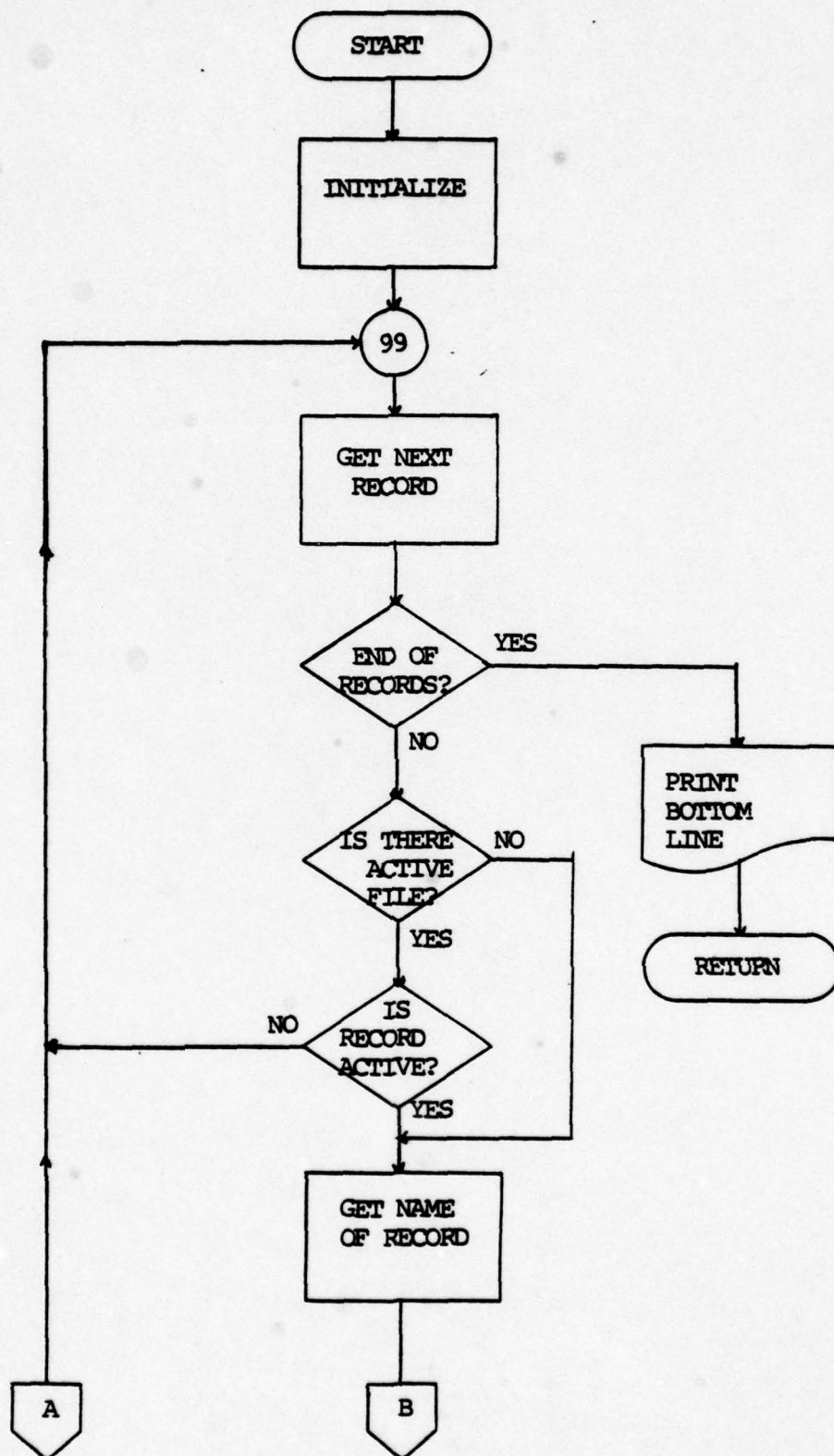
(1) CODE, MASK, ARG2, ARG4, NROWS, NCOL, IER as described previously.

(2) K1, K2, K3 : Record number to start searching, to stop searching and searching step respectively. Default values are 1, (NROWS + NCOL), 1.

(3) IFLG : Flag to determine the data structure type of the packed file. If IFLG=0, data structure SPARSE otherwise SUPERSPARSE.

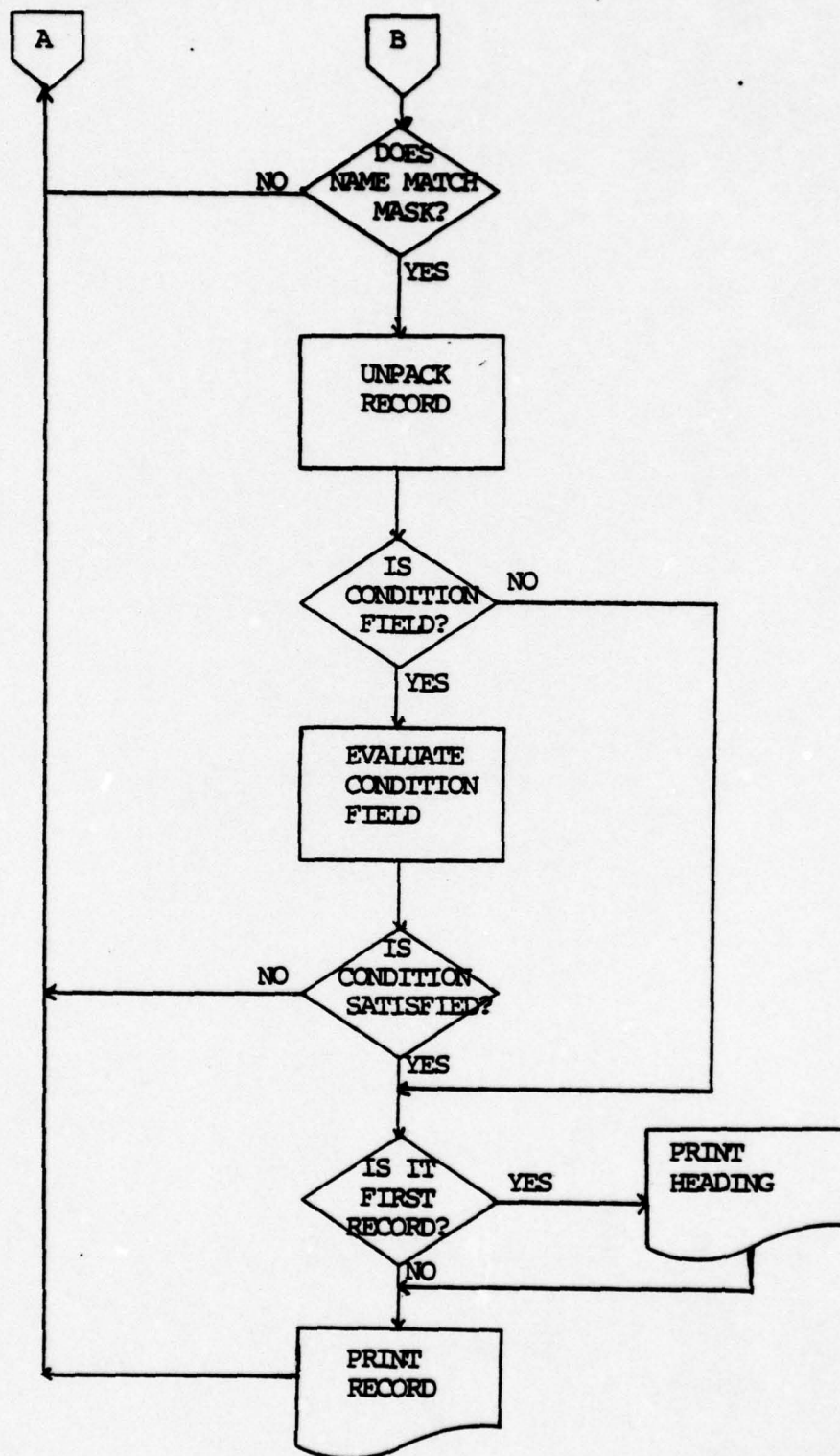
(4) IACT : Passed through the INIT common area and, if 0, the searching is done as determined by K1, K2, K3, otherwise there is an active file created by the user through the ACTIVATE command and only the members of this file are searched.

The logic of this subroutine is shown in the flowchart on the next two pages.



INTERPRETER





# COMPUTER PROGRAM LISTING

```

*****
GALATAS - MARCH 1979 - HELLENIC ARMY
*****
NAME OF THE SYSTEM : ATHENA
LANGUAGE : FORTRAN G
*****
SYSTEM ATHENA IS USED FOR HANDLING A LARGE SCALE LINEAR
PROGRAMMING SOLUTION FILE.
USES AS INPUT A SOLUTION FILE IN STANDARD FORMAT AND THROUGH
A SET OF QUERIES HELP THE ANALYST TO GET INTERACTIVELY THE
SPECIFIC INFORMATION HE NEEDS TO ANALYZE THE SOLUTION WITHOUT
GOING THROUGH THE STACK OF COMPUTER OUTPUT. A SOPHISTICATED
THE SYSTEM PACKS THE SOLUTION FILE INTO A SOPHISTICATED
DATA STRUCTURE FOR MEMORY SPACE ECONOMY.
TWO KIND OF DATA STRUCTURES ARE EMPLOYED:
1. FOR GENERAL TYPE SOLUTION FILE
2. FOR SOLUTION FILES WHERE THE NUMBER OF DISTINCT VALUES
IS RELATIVELY SMALL (LESS THAN ONE HALF OF THE TOTAL
NUMBERS STORED.)
*****
FOR MORE INFORMATION ABOUT ACCEPTABLE QUERIES, DATA
STRUCTURES AND USE OF THE SYSTEM SEE USER'S MANUAL.
*****
M A I N P R O G R A M
*****
COPYRIGHT 1979
PAN. GALATAS, GREEK ARMY HEADQUARTERS, ATHENS, GREECE
ALL RIGHTS RESERVED
REPRODUCTION OF ALL, OR ANY PART OF SYSTEM ATHENA
IS PROHIBITED WITHOUT WRITTEN PERMISSION OF THE AUTHOR.
*****
DECLARATIONS
*****
COMMON I,M(6),TIME
INTEGER*2 QRY(80),MASK(8),CODE(20)
COMMON /INIT/ZERO,TIME,IACT,MEMOR,ISIZE
COMMON /SOLPAC/SOLFIL(81920)
COMMON /CHAR/A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
- BLANK,COMMA,STAR,LPAR,RPAR
INTEGER*2 A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
- BLANK,COMMA,STAR,LPAR,RPAR
*****

```

MAI00010  
MAI00020  
MAI00030  
MAI00040  
MAI00050  
MAI00060  
MAI00070  
MAI00080  
MAI00090  
MAI00100  
MAI00110  
MAI00120  
MAI00130  
MAI00140  
MAI00150  
MAI00160  
MAI00170  
MAI00180  
MAI00190  
MAI00200  
MAI00210  
MAI00220  
MAI00230  
MAI00240  
MAI00250  
MAI00260  
MAI00270  
MAI00280  
MAI00290  
MAI00300

MAI00310  
MAI00320  
MAI00330  
MAI00340  
MAI00350  
MAI00360  
MAI00370  
MAI00380  
MAI00390  
MAI00400  
MAI00410  
MAI00420  
MAI00430

CC

MAI00440  
MAI00450  
MAI00460  
MAI00470  
MAI00480

REAL ARG2,ARG4  
LOGICAL\*1,VER, PROM,HEAD  
DATA KK//NO  
DEFINE FILE 4(27,1775,U,IJ)  
DEFINE FILE 3(32,1775,U,JJ)

OBJECT CODE COPYRIGHT STATEMENT

IF(.FALSE.) WRITE(6,9999)  
FORMAT('COPYRIGHT 1979,  
1 PAN. GALATAS, GREEK ARMY HEADQUARTERS, ATHENS, GREECE',/,  
2 'ALL RIGHTS RESERVED.')

MEMOR=81920  
CALL IXCLOCK(ITM)  
TIME=ITM(5)/76800.

GET INFORMATION NEEDED FOR THE SOLUTION FILE

1004 WRITE(6,1004)  
FORMAT(2X,'WHAT IS THE FILE NO OF THE SOLUTION FILE ? (FORMAT I2)

1005 READ(5,1005) IFLR  
FORMAT(I2)

1001 WRITE(6,1001)  
FORMAT(2X,'IS THE FILE ALREADY PACKED ? ENTER YES OR NO :')

1002 READ(5,1002) K1  
FORMAT(A4)

IF(K1.NE.KK) GO TO 1099

1003 WRITE(6,1003)  
FORMAT(2X,'FOR BETTER MEMORY UTILIZATION ANSWER YES OR NO TO THE  
FOLLOWING QUESTION : '//IX,'DO YOU EXPECT THE DISTINCT VALUES OF  
THE SOLUTION FILE TO BE LESS THAN HALF OF THOSE NUMBERS EXCLUDING  
ZEROS AND INFINITY ? :')

1007 READ(5,1002) K1  
WRITE(6,1007)  
FORMAT(5X,'WHAT IS THE FILE NO YOU WANT THE PACKED FILE TO BE WRIT

1006 TEN ? (FORMAT I2),')

1006 READ(5,1005) IFLW  
WRITE(6,1006)  
FORMAT(10X,'READING AND PACKING THE SOLUTION FILE')

READING AND PACKING THE SOLUTION FILE

IF(K1.EQ.KK) CALL READF(NROWS,NCOL,IER,IFLR,IFLW)  
IF(K1.NE.KK) CALL READD(NROWS,NCOL,IER,IFLR,IFLW)

MAI00490  
MAI00500  
MAI00510  
MAI00520  
MAI00530  
MAI00540  
MAI00550  
MAI00560  
MAI00570  
MAI00580  
MAI00590  
MAI00600  
MAI00610  
MAI00620  
MAI00630  
MAI00640  
MAI00650  
MAI00660  
MAI00670  
MAI00680  
MAI00690  
MAI00700  
MAI00710  
MAI00720  
MAI00730  
MAI00740  
MAI00750  
MAI00760  
MAI00770  
MAI00780  
MAI00790  
MAI00800  
MAI00810  
MAI00820  
MAI00830



MAI00840  
MAI00850  
MAI00860  
MAI00870  
MAI00880  
MAI00890  
MAI00900  
MAI00910  
MAI00920  
MAI00930  
MAI00940  
MAI00950  
MAI00960  
MAI00970  
MAI00980  
MAI00990  
MAI01000  
MAI01010  
MAI01020  
MAI01030  
MAI01040  
MAI01050  
MAI01060  
MAI01070  
MAI01080  
MAI01090  
MAI01100  
MAI01110  
MAI01120  
MAI01130  
MAI01140  
MAI01150  
MAI01160  
MAI01170  
MAI01180  
MAI01190  
MAI01200  
MAI01210  
MAI01220  
MAI01230  
MAI01240  
MAI01250  
MAI01260  
MAI01270  
MAI01280  
MAI01290  
MAI01300  
MAI01310

```

1099 IF( IER, NE, 0 ) GO TO 999
      GO TO 1020
      CONTINUE
      READ( IFLR, 1 ) ( SOLFIL( IK ), IK=1, 100 )
      IK= SOLFIL( 1 )
      WRITE( 6, 1066 ) IK
1066 FORMAT( 5X, 'MEMORY REQUIREMENTS FOR THE PACKED FILE :', I5, ' 4-BYTE
      -WORDS: ', //, 3X, 'IF YOU HAVE IT ENTER YES, //, 3X, 'OTHERWISE ENTER NO
      - MAKE ADJUSTMENTS AT COMMON AND TRY AGAIN' )
      READ( 5, 1002 ) K1
      IF( K1, EQ, KK ) GO TO 99
      READ( IFLR, 1 ) ( SOLFIL( IL ), IL=1, IK )
1020 NROWS= SOLFIL( 3 )
      NCOL= SOLFIL( 4 )
      ISIZE= SOLFIL( 1 )
      IFLG= SOLFIL( 2 )

C      INITIALIZE FLAGS - CALCULATE LIMITS
C
C
C
92  IER=0
      VER=.FALSE.
      PROM=.TRUE.
      HEAD=.FALSE.
      K1=1
      K2=NROWS+NCOL
      K3=1
10  CONTINUE
      DO 2 J=1, 18
2  CODE( J )=0
      CALL TIMER
      IF( PROM ) WRITE( 6, 100 )
      FORMAT( 15X, ' INPUT A COMMAND ' )
100  READ THE QUERY
C
C
C      READ( 5, 200 ) QRY
C
C      CHECK IF IT IS HEADING
C
C      IF( QRY( 1 ), EQ, H ) GO TO 61
      IF( VER ) WRITE( 6, 300 ) QRY
      FORMAT( 80A1 )
200  CALL THE PARSER TO PARSE THE QUERY
C
C
C      CALL PARSE( QRY, CODE, MASK, ARG2, ARG4, ICOL1, IER )

```







PAR00210  
PAR00220  
PAR00230  
PAR00240  
PAR00250  
PAR00260  
PAR00270  
PAR00280  
PAR00290  
PAR00300  
PAR00310  
PAR00320  
PAR00330  
PAR00340  
PAR00350  
PAR00360  
PAR00370  
PAR00380  
PAR00390  
PAR00400  
PAR00410  
PAR00420  
PAR00430  
PAR00440  
PAR00450  
PAR00460  
PAR00470  
PAR00480  
PAR00490  
PAR00500  
PAR00510  
PAR00520  
PAR00530  
PAR00540  
PAR00550  
PAR00560  
PAR00570  
PAR00580  
PAR00590  
PAR00600  
PAR00610  
PAR00620  
PAR00630  
PAR00640  
PAR00650  
PAR00660  
PAR00670  
PAR00680

```

C IF COMMENTS IGNORE THE QUERY
C
C IF (QRY(1).NE.STAR) GO TO 10
C CODE(1)=10
C RETURN
C DO 1 J=1,80
C IF (QRY(J).EC.BLANK) GO TO 1
C ICOL1=J
C GO TO 20
C CONTINUE
C IER=1
C RETURN
C
C PRJESS THE TYPE FIELD OF THE COMMAND
C
C CALL TYPE(QRY,ICOL1,CODE,IER)
C IF (IER.NE.0) RETURN
C
C IF CONTROL TYPE COMMAND RETURN OTHERWISE EXAMINE
C THE REMAINING FIELDS.
C
C IF (CODE(1).LT.10) RETURN
C CALL GETNCH(QRY,ICOL1,BLANK,IER)
C IF (IER.NE.0) RETURN
C CALL QJAL(QRY,ICOL1, CODE, IER)
C IF (IER.NE.0) RETURN
C CALL GETNCH(QRY,ICOL1,BLANK,IER)
C IF (IER.NE.0) AND CODE(1).NE.17 AND CODE(2).NE.1) RETURN
C IF (IER.EQ.0) GO TO 21
C IER=0
C RETURN
C CALL MASKA(QRY,ICOL1, CODE, MASK, STAR, BLANK, IER)
C IF (IER.NE.0) RETURN
C CALL GETNCH(QRY,ICOL1,BLANK,IER)
C IF (IER.EQ.0) GO TO 30
C IF (IER.EQ.201) IER=0
C RETURN
C CALL COND(QRY,ICOL1, CODE, ARG2, ARG4, IER)
C IF (IER.EQ.501) GO TO 40
C IF (IER.NE.0) RETURN
C CALL GETNCH(QRY,ICOL1,BLANK,IER)
C IF (IER.EQ.0) GO TO 40
C IF (IER.EQ.201) IER=0
C RETURN
C CALL PRINT(QRY,ICOL1, CODE, IER)
C RETURN
C END

```

PAR00690  
PAR00700  
PAR00710  
PAR00720  
PAR00730  
PAR00740  
PAR00750  
PAR00760  
PAR00770  
PAR00780  
PAR00790  
PAR00800  
PAR00810  
PAR00820  
PAR00830  
PAR00840  
PAR00850  
PAR00860  
PAR00870  
PAR00880  
PAR00890  
PAR00900  
PAR00910  
PAR00920  
PAR00930  
PAR00940  
PAR00950  
PAR00960  
PAR00970  
PAR00980  
PAR00990  
PAR01000  
PAR01010  
PAR01020  
PAR01030  
PAR01040  
PAR01050  
PAR01060  
PAR01070  
PAR01080  
PAR01090  
PAR01100  
PAR01110  
PAR01120  
PAR01130  
PAR01140  
PAR01150  
PAR01160

```

*****
** SUBROUTINE TYPE PARSSES THE TYPE FIELD OF THE COMMAND
**
**      INTERNAL      CODES
**      -----
**      1
**      2
**      3
**      4
**      5
**      6
**      7
**
**      11
**      12
**      13
**      14
**      15
**      16
**      17
**
*****

```

```

SUBROUTINE TYPE (QRY, ICOL1, CODE, IER)
INTEGER *2 QRY(80), CODE(20)
COMMON /CHAR/A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
-BLANK,COMMA,STAR,LPAR,RPAR
INTEGER *2 A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
-BLANK,COMMA,STAR,LPAR,RPAR
IER=0
IF(QRY(ICOL1).NE.D) GO TO 10
CODE(1)=11
IF(QRY(ICOL1+1).EQ.E) CODE(1)=17
RETURN
IF(QRY(ICOL1).NE.C) GO TO 20
CODE(1)=12
RETURN
IF(QRY(ICOL1).NE.A) GO TO 30
IF(QRY(ICOL1+1).NE.V) GO TO 25
CODE(1)=14
RETURN
CODE(1)=13
IF(QRY(ICOL1+1).EQ.D.OR.QRY(ICOL1+1).EQ.BLANK) RETURN
CODE(1)=16
IF(QRY(ICOL1+1).EQ.C) RETURN

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

10

20

25

PAR01170  
PAR01180  
PAR01190  
PAR01200  
PAR01210  
PAR01220  
PAR01230  
PAR01240  
PAR01250  
PAR01260  
PAR01270  
PAR01280  
PAR01290  
PAR01300  
PAR01310  
PAR01320  
PAR01330  
PAR01340  
PAR01350  
PAR01360  
PAR01370  
PAR01380  
PAR01390  
PAR01400  
PAR01410  
PAR01420  
PAR01430  
PAR01440  
PAR01450  
PAR01460  
PAR01470  
PAR01480  
PAR01490  
PAR01500  
PAR01510  
PAR01520  
PAR01530  
PAR01540  
PAR01550  
PAR01560  
PAR01570  
PAR01580  
PAR01590  
PAR01600  
PAR01610  
PAR01620  
PAR01630  
PAR01640

```

30  IER=101
    RETURN
    IF (QRY(ICOL1),NE.S) GO TO 40
    CODE(1)=15
    IF (QRY(ICOL1+1).EQ.O.OR.QRY(ICOL1+1).EQ.BLANK) RETURN
    CODE(1)=7
    IF (QRY(ICOL1+1).EQ.E.AND.QRY(ICOL1+2).EQ.T) RETURN
    IER=102
    RETURN
    IF (QRY(ICOL1),NE.V) GO TO 50
    CODE(1)=1
    RETURN
    IF (QRY(ICOL1),NE.P) GO TO 60
    CODE(1)=3
    RETURN
    IF (QRY(ICOL1),NE.H) GO TO 70
    CODE(1)=5
    RETURN
    IF (QRY(ICOL1),NE.N.AND.QRY(ICOL1+1).NE.O) GO TO 80
    CODE(1)=2
    IF (QRY(ICOL1+2).EQ.V) RETURN
    CODE(1)=4
    IF (QRY(ICOL1+2).EQ.P) RETURN
    IER=103
    RETURN
    CODE(1)=6
    IF (QRY(ICOL1).EQ.E) RETURN
    IER=104
    RETURN
    END
80

```

```

*****
* SUBROUTINE QVAL PARSSES THE QUALIFIED FIELD
*          INTERNAL CODES
*          -----
*          A-LL : 1
*          C-OLUMNS : 2
*          R-OWS : 3
*****

```

```

SUBROUTINE QVAL(QRY,ICOL1,CODE,IER)
INTEGER*2 QRY(80),CODE(20)
COMMON /CHAR/A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
-BLANK,COMMA,STAR,LPAR,RPAR

```



```

INTEGER*2 A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
-BLANK,COMMA,STAR,LPAR,RPAR
IER=0
IF(QRY(ICOL1).NE.A) GO TO 10
CODE(2)=1
RETURN
IF(QRY(ICOL1).NE.C) GO TO 20
CODE(2)=2
RETURN
IF(QRY(ICOL1).NE.R) IER=301
CODE(2)=3
RETURN
END

```

10

20

CCCCCCCCCCCC

```

*****
* SUBROUTINE MASKA PROCESS THE MASK FIELD OF THE COMMAND
*
* INTERNAL
*
*-----
* MASK ALL * : 1
* MASK NOT ALL * : 2
* MASK WITHOUT ANY * : 3
*
*****

```

```

SUBROUTINE MASKA(QRY,ICOL1,CODE,MASK,STAR,BLANK,IER)
INTEGER*2 QRY(1),CODE(1),MASK(1),STAR,BLANK
IFLAG1=0
IFLAG2=0
DO 1 I=1,8
  MASK(I)=STAR
  IF(IFLAG1.NE.0) GO TO 1
  IF(QRY(ICOL1+I-1).NE.BLANK) GO TO 10
  IFLAG1=1
  GO TO 1
  IF(QRY(ICOL1+I-1).NE.STAR) IFLAG2=IFLAG2+1
  MASK(I)=QRY(ICOL1+I-1)
  CCNTINUE
CODE(3)=2
IF(IFLAG2.EQ.0) CODE(3)=1
IF(IFLAG2.EQ.8) CODE(3)=3
RETURN
END

```

10

1

CCC

PAR01650  
PAR01660  
PAR01670  
PAR01680  
PAR01690  
PAR01700  
PAR01710  
PAR01720  
PAR01730  
PAR01740  
PAR01750  
PAR01760  
PAR01770  
PAR01780  
PAR01790  
PAR01800  
PAR01810  
PAR01820  
PAR01830  
PAR01840  
PAR01850  
PAR01860  
PAR01870  
PAR01880  
PAR01890  
PAR01900  
PAR01910  
PAR01920  
PAR01930  
PAR01940  
PAR01950  
PAR01960  
PAR01970  
PAR01980  
PAR01990  
PAR02000  
PAR02010  
PAR02020  
PAR02030  
PAR02040  
PAR02050  
PAR02060  
PAR02070  
PAR02080  
PAR02090  
PAR02100  
PAR02110  
PAR02120

PAR02130  
PAR02140  
PAR02150  
PAR02160  
PAR02170  
PAR02180  
PAR02190  
PAR02200  
PAR02210  
PAR02220  
PAR02230  
PAR02240  
PAR02250  
PAR02260  
PAR02270  
PAR02280  
PAR02290  
PAR02300  
PAR02310  
PAR02320  
PAR02330  
PAR02340  
PAR02350  
PAR02360  
PAR02370  
PAR02380  
PAR02390  
PAR02400  
PAR02410  
PAR02420  
PAR02430  
PAR02440  
PAR02450  
PAR02460  
PAR02470  
PAR02480  
PAR02490  
PAR02500  
PAR02510  
PAR02520  
PAR02530  
PAR02540  
PAR02550  
PAR02560  
PAR02570  
PAR02580  
PAR02590  
PAR02600

SUBROUTINE COND PARSES THE CONDITION FIELD OF THE  
QUERY AND GENERATES THE CORRESPONDING CODE.  
THERE ARE 3 KINDS OF SIMPLE CONDITIONAL PHRASES :

1. <ARG1> <RELOP> <ARG2>  
INTERNAL CODES

ARG1	RELOP
1	1
2	2
3	3
4	4
5	5
6	6
7	7

ARG2

A. AS ARG1 OR INTEGER NUMBER  
B. ANY REAL OR INTEGER NUMBER  
2. <ARG1> <MIN> OR <ARG1> <MAX>  
INTERNAL CODES

MIN	7
MAX	8

3. STATUS <ARG> OR ST <ARG>  
INTERNAL CODES

STATUS OR ST : 9  
ARG

IN	00	FC	07	TR	14
BS	01	BC	08	PP	15
LL	02	VR	09	IE	16
UL	03	SR	10	AE	17
EQ	04	RR	11	CE	18
VC	05	FR	12	DE	19
SC	06	ER	13		

THERE ARE 2 KINDS OF COMPOUND CONDITIONAL PHRASES :

1. <1ST SIMP> <RELOP> <1ST SIMP>  
2. <1ST SIMP> <RELOP> <2ND SIMP>  
INTERNAL CODES

OR	1
AND	2

FOR MORE DETAILS SEE USER'S MANUAL

\*\*\*\*\*

CC

```

SUBROUTINE COND(QRY, ICOLL, CODE, ARG2, ARG4, IER)
  INTEGER*2 QRY(80), CODE(20)
  COMMON /CHAR/A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
  -BLANK,COMMA,STAR,LPAR,RPAR
  INTEGER*2 A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
  -BLANK,COMMA,STAR,LPAR,RPAR
  REAL ARG2, ARG4
  IER=0

  CHECK IF WORD 'FOR' EXIST
    IF(QRY(ICOLL),EQ.F) GO TO 10
    IER=501
    RETURN
  10 IF(QRY(ICOLL+1).EQ.O.AND.QRY(ICOLL+2).EQ.R) GO TO 20
    IER=502
    RETURN
  20 ICOLL=ICOLL+3
    IF(QRY(ICOLL).EQ.BLANK) CALL GETNCH(QRY,ICOLL,BLANK,IER)
    IF(IER.NE.0) RETURN
    C
    C
    C
    CHECK FOR LEFT PARENTHESIS
    IF(QRY(ICOLL).EQ.LPAR) GO TO 30
    IER=503
    RETURN
  30 ICOLL=ICOLL+1
    IF(QRY(ICOLL).EQ.BLANK) CALL GETNCH(QRY,ICOLL,BLANK,IER)
    IF(IER.NE.0) RETURN
    C
    C
    C
    IF 1ST KIND OF CONDITIONAL PHRASE CALL SIMCOD
    INDEX=4
    IF(QRY(ICOLL).EQ.S.AND.QRY(ICOLL+1).EQ.T) GO TO 40
    CALL SIMCOD(QRY,ICOLL,CODE,ARG2,INDEX,IER)
    IF(IER.NE.0) RETURN
    D) 1 J=1,15
    IF(QRY(ICOLL+J).NE.BLANK) GO TO 1
    ICOLL=ICOLL+J-1
    GO TO 50
  1 CONTINUE
    IER=504
    RETURN
  50 IF(QRY(ICOLL).EQ.RPAR) RETURN
    CALL GETNCH(QRY,ICOLL,BLANK,IER)
    IF(IER.NE.0) RETURN

```

PAR02610  
 PAR02620  
 PAR02630  
 PAR02640  
 PAR02650  
 PAR02660  
 PAR02670  
 PAR02680  
 PAR02690  
 PAR02700  
 PAR02710  
 PAR02720  
 PAR02730  
 PAR02740  
 PAR02750  
 PAR02760  
 PAR02770  
 PAR02780  
 PAR02790  
 PAR02800  
 PAR02810  
 PAR02820  
 PAR02830  
 PAR02840  
 PAR02850  
 PAR02860  
 PAR02870  
 PAR02880  
 PAR02890  
 PAR02900  
 PAR02910  
 PAR02920  
 PAR02930  
 PAR02940  
 PAR02950  
 PAR02960  
 PAR02970  
 PAR02980  
 PAR02990  
 PAR03000  
 PAR03010  
 PAR03020  
 PAR03030  
 PAR03040  
 PAR03050  
 PAR03060  
 PAR03070  
 PAR03080



PAR03090  
PAR03100  
PAR03110  
PAR03120  
PAR03130  
PAR03140  
PAR03150  
PAR03160  
PAR03170  
PAR03180  
PAR03190  
PAR03200  
PAR03210  
PAR03220  
PAR03230  
PAR03240  
PAR03250  
PAR03260  
PAR03270  
PAR03280  
PAR03290  
PAR03300  
PAR03310  
PAR03320  
PAR03330  
PAR03340  
PAR03350  
PAR03360  
PAR03370  
PAR03380  
PAR03390  
PAR03400  
PAR03410  
PAR03420  
PAR03430  
PAR03440  
PAR03450  
PAR03460  
PAR03470  
PAR03480  
PAR03490  
PAR03500  
PAR03510  
PAR03520  
PAR03530  
PAR03540  
PAR03550  
PAR03560

IF(QRY(ICOL1).EQ.RPAR) RETURN  
IER=505  
IF(CODE(5).EQ.7.OR.CODE(5).EQ.8) RETURN  
IER=0

C  
C  
C

FIND WHAT RELOP (OR , AND) EXIST

IF(QRY(ICOL1).NE.0) GO TO 60  
CODE(7)=1  
IF(QRY(ICOL1+1).EQ.R) GO TO 61  
IER=506  
RETURN

IF(QRY(ICOL1).NE.A) GO TO 70  
CODE(7)=2

60

IF(QRY(ICOL1+1).EQ.N.AND.QRY(ICOL1+2).EQ.D) GO TO 61

IER=507  
RETURN

IER=508  
RETURN

70

CALL GETNCH(QRY,ICOL1,BLANK,IER)  
IF(IER.NE.0) RETURN  
INDEX=8

61

IF COMPOUNT COMDITON CALL SIMCOD

IF(QRY(ICOL1).EQ.S.AND.QRY(ICOL1+1).EQ.T) GC TO 40  
CALL SIMCOD(QRY,ICOL1,CODE,ARG4,INDEX,IER)  
IF(IER.NE.0) RETURN  
GO TO 301

C  
C  
C

PARSE THE 2ND KIND OF CONDITION. FIND THE OPERAND OF  
STATUS

C  
C  
C  
C  
40

CCODE(INDEX)=9  
CALL GETNCH(QRY,ICOL1,BLANK,IER)  
IF(IER.NE.0) RETURN  
IF(QRY(ICOL1).NE.8) GO TO 110  
CODE(INDEX+1)=1  
IF(QRY(ICOL1+1).EQ.S) GO TO 301  
IF(QRY(ICOL1+1).EQ.C) GO TO 143  
IER=511  
RETURN

110

IF(QRY(ICOL1).NE.L) GO TO 120  
CODE(INDEX+1)=2  
IF(QRY(ICOL1+1).EQ.L) GO TO 301

```

120      IER=512
        RETURN
        IF (ORY (ICOL1).NE.U) GO TO 130
        CODE (INDEX+1)=3
        IF (ORY (ICOL1+1).EQ.L) GO TO 301
        IER=513
        RETURN
        IF (ORY (ICOL1).NE.E) GO TO 140
        CODE (INDEX+1)=4
        IF (ORY (ICOL1+1).EQ.Q) GO TO 301
        IF (ORY (ICOL1+1).EQ.R) GO TO 131
        IER=514
        RETURN
        IF (ORY (ICOL1+1).NE.C) GO TO 150
        IF (ORY (ICOL1).NE.V) GO TO 141
        CODE (INDEX+1)=5
        GO TO 301
        IF (ORY (ICOL1).NE.S) GO TO 142
        CODE (INDEX+1)=6
        GO TO 301
        IF (ORY (ICOL1).NE.F) GO TO 143
        CODE (INDEX+1)=7
        GO TO 301
        CODE (INDEX+1)=8
        IF (ORY (ICOL1).EQ.B) GO TO 301
        IER=515
        RETURN
        IF (ORY (ICOL1+1).NE.R) GO TO 160
        IF (ORY (ICOL1).NE.V) GO TO 151
        CODE (INDEX+1)=9
        GO TO 301
        IF (ORY (ICOL1).NE.S) GO TO 152
        CODE (INDEX+1)=10
        GO TO 301
        IF (ORY (ICOL1).NE.R) GO TO 153
        CODE (INDEX+1)=11
        GO TO 301
        IF (ORY (ICOL1).NE.F) GO TO 154
        CODE (INDEX+1)=12
        GO TO 301
        IF (ORY (ICOL1).NE.E) GO TO 155
        CODE (INDEX+1)=13
        GO TO 301
        CODE (INDEX+1)=14
        IF (ORY (ICOL1).EQ.T) GO TO 301
        IER=516
        RETURN
        IF (ORY (ICOL1+1).NE.P) GO TO 170

```

```

PAR03570
PAR03580
PAR03590
PAR03600
PAR03610
PAR03620
PAR03630
PAR03640
PAR03650
PAR03660
PAR03670
PAR03680
PAR03690
PAR03700
PAR03710
PAR03720
PAR03730
PAR03740
PAR03750
PAR03760
PAR03770
PAR03780
PAR03790
PAR03800
PAR03810
PAR03820
PAR03830
PAR03840
PAR03850
PAR03860
PAR03870
PAR03880
PAR03890
PAR03900
PAR03910
PAR03920
PAR03930
PAR03940
PAR03950
PAR03960
PAR03970
PAR03980
PAR03990
PAR04000
PAR04010
PAR04020
PAR04030
PAR04040

```





```

C *****
SUBROUTINE SIMCOD(QRY,ICOL1,CODE,ARG2,INDEX,IER)
  INTEGER*2 QRY(80),CODE(20)
  COMMON /CHAR/A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
  -BLANK,COMMA,STAR,LPAR,RPAR
  INTEGER*2 A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q,R,S,T,U,V,W,X,
  -BLANK,COMMA,STAR,LPAR,RPAR
  REAL ARG2,ARG4
  IER=0
  IFLAG=0

  FINE ARG1 AND ARG2.

  IF(QRY(ICOL1).NE.X) GO TO 20
  CODE(INDEX+IFLAG)=1
  GO TO 100
  IF(QRY(ICOL1).NE.S.AND.QRY(ICOL1).NE.C) GO TO 30
  CODE(INDEX+IFLAG)=2
  GO TO 100
  IF(QRY(ICOL1).NE.L) GO TO 40
  CODE(INDEX+IFLAG)=3
  GO TO 100
  IF(QRY(ICOL1).NE.U) GO TO 50
  CODE(INDEX+IFLAG)=4
  GO TO 100
  IF(QRY(ICOL1).NE.D) GO TO 60
  CODE(INDEX+IFLAG)=5
  GO TO 100
  IF(QRY(ICOL1).NE.P) GO TO 70
  CODE(INDEX+IFLAG)=6
  GO TO 100
  IF(QRY(ICOL1).NE.W) GO TO 80
  CODE(INDEX+IFLAG)=7
  GO TO 100
  IF(FLAG.NE.0) GO TO 90
  IER=601
  RETURN
  DO 1 J=1,12
  IF(QRY(ICOL1+J).NE.BLANK) GO TO 1
  ICOL2=ICOL1+J-1
  GO TO 91
  CONTINUE
  IER=602
  RETURN
  IF(QRY(ICOL2).EQ.RPAR) ICOL2=ICOL2-1
  ARG2=VAL(QRY,ICOL1,ICOL2)
  CODE(INDEX+2)=9
  RETURN

```

```

PAR04530
PAR04540
PAR04550
PAR04560
PAR04570
PAR04580
PAR04590
PAR04600
PAR04610
PAR04620
PAR04630
PAR04640
PAR04650
PAR04660
PAR04670
PAR04680
PAR04690
PAR04700
PAR04710
PAR04720
PAR04730
PAR04740
PAR04750
PAR04760
PAR04770
PAR04780
PAR04790
PAR04800
PAR04810
PAR04820
PAR04830
PAR04840
PAR04850
PAR04860
PAR04870
PAR04880
PAR04890
PAR04900
PAR04910
PAR04920
PAR04930
PAR04940
PAR04950
PAR04960
PAR04970
PAR04980
PAR04990
PAR05000

```

PAR05010  
PAR05020  
PAR05030  
PAR05040  
PAR05050  
PAR05060  
PAR05070  
PAR05080  
PAR05090  
PAR05100  
PAR05110  
PAR05120  
PAR05130  
PAR05140  
PAR05150  
PAR05160  
PAR05170  
PAR05180  
PAR05190  
PAR05200  
PAR05210  
PAR05220  
PAR05230  
PAR05240  
PAR05250  
PAR05260  
PAR05270  
PAR05280  
PAR05290  
PAR05300  
PAR05310  
PAR05320  
PAR05330  
PAR05340  
PAR05350  
PAR05360  
PAR05370  
PAR05380  
PAR05390  
PAR05400  
PAR05410  
PAR05420  
PAR05430  
PAR05440  
PAR05450  
PAR05460  
PAR05470  
PAR05480

```

C      FIND RELATIONAL OPERATOR (GT, EQ, E.T.C.)
C100  IF (IFLAG.NE.0) RETURN
      IFLAG=2
      CALL GETNCH(QRY,ICOLL,BLANK,IER)
      IF (IER.NE.0) RETURN
      IF (QRY(ICOLL).NE.G) GO TO 11
      IF (QRY(ICOLL+1).NE.T) GO TO 12
      CODE(INDEX+1)=3
      GO TO 101
      CODE(INDEX+1)=4
      IF (QRY(ICOLL+1).EQ.E) GO TO 101
      IER=603
      RETURN
      IF (QRY(ICOLL).NE.L) GO TO 21
      IF (QRY(ICOLL+1).NE.T) GO TO 22
      CODE(INDEX+1)=5
      GO TO 101
      CODE(INDEX+1)=6
      IF (QRY(ICOLL+1).EQ.E) GO TO 101
      IER=604
      RETURN
      IF (QRY(ICOLL).NE.E) GO TO 31
      CODE(INDEX+1)=1
      IF (QRY(ICOLL+1).EQ.O) GO TO 101
      IER=605
      RETURN
      IF (QRY(ICOLL).NE.N) GO TO 41
      CODE(INDEX+1)=2
      IF (QRY(ICOLL+1).EQ.E) GO TO 101
      IER=606
      RETURN
      IF (QRY(ICOLL).NE.M) GO TO 51
      IF (QRY(ICOLL+1).NE.I) GO TO 52
      CODE(INDEX+1)=7
      IF (QRY(ICOLL+2).EQ.N) RETURN
      IER=607
      RETURN
      CODE(INDEX+1)=8
      IF (QRY(ICOLL+1).EQ.A.AND.QRY(ICOLL+2).EQ.X) RETURN
      IER=608
      RETURN
      IER=609
      RETURN
      CALL GETNCH(QRY,ICOLL,BLANK,IER)
      IF (IER.EQ.0) GO TO 99
      RETURN
      END
101
51
52
41
31
21
22
11
12

```







PAR06450  
PAR06460  
PAR06470  
PAR06480  
PAR06490  
PAR06500  
PAR06510  
PAR06520  
PAR06530  
PAR06540  
PAR06550  
PAR06560  
PAR06570  
PAR06580  
PAR06590  
PAR06600  
PAR06610  
PAR06620  
PAR06630  
PAR06640  
PAR06650  
REA00010  
REA00020  
REA00030  
REA00040  
REA00050  
REA00060  
REA00070  
REA00080  
REA00090  
REA00100  
REA00110  
REA00120  
REA00130  
REA00140  
REA00150  
REA00160  
REA00170  
REA00180  
REA00190  
REA00200  
REA00210  
REA00220  
REA00230  
REA00240  
REA00250  
REA00260  
REA00270

```

MET=0
DO 1 I=1ST, IED
  IF(X(I).EQ.MINUS) GO TO 10
  IF(X(I).EQ.BLANK) GO TO 1
  MET=MET+1
  IF(X(I).NE.DECIM) GO TO 20
  IEXP=MET
  GO TO 1
  ITEMP=X(I)-ZERO
  ITEMP=ITEMP/LL
  WRITE(6,100) IFN, ITEMP
  FORMAT(5X, I10, I10)
  IFN=IFN*10+ITEMP
  GO TO 1
  SIGN=-1.
  CONTINUE
  IEXP=MET-IEXP
  IF(IEXP.EQ.0) LEXP=0
  VAL=SIGN*IFN*10.**(-LEXP)
  RETURN
END
*****
SUBROUTINE READF ACCEPTS AS INPUT A FILE IN 'STANDARD'
FORMAT WITH FILES NUMBER 'IFLR', PACKS IT IN SPARSE DATA
STRUCTURE, WRITES IT IN A DIRECT ACCESS FILE 'IFLW' IN PACKED
FORM AND RETURNS THE PACKED FILE IN 'SOLFIL' WITH INFORMATIONS
ABOUT THE SIZE OF FILE:
  SOLFIL(1) = SIZE OF FILE IN 4-BYTE WORDS
  SOLFIL(2) = 0 (DATA STRUCTURE SPARSE)
  SOLFIL(3) = NUMBER OF ROWS
  SOLFIL(4) = NUMBER OF COLUMNS
*****
SUBROUTINE READF(NROWS, NCOL, IER, IFLR, IFLW)
  REAL DATA(7), SOLIST(1)
  INTEGER*2 CSTAT(15), SOFLAG(1), STAT
  LOGICAL*1 BITS(16)
  COMMON /INIT/ZERO, TINF, IACT, MEMOR
  COMMON /SOLPAC/SOLFIL(81920)
  EQUIVALENCE (SOLFIL(5), SJFLAG(1), SOLIST(1))
  DATA CSTAT/2HIN, 2HBS, 2HLL, 2HUL, 2HEQ, 2HVC, 2HSC, 2HFC, 2HBC,
  -2HVR, 2HHR, 2HFR, 2HER, 2HTR/
  IER=0
  SEE TAPSET SUBROUTINE IN CP/CMS MANUAL.

```

20  
C 100  
10  
1  
CCCCCCCCCCCCCCCC  
C C C

```

C      CALL TAPSET(2,IFLR,80,17,1,80)
C
C      READ IN NR0WS, NCOL AND IFLP
C      IF IFLP=0 THERE ARE NOT PENALTIES IN THE FILE
C
100    READ(IFLR,100) NR0WS,NCOL,IFLGP
        FORMAT(15,30X,15,10X,11)
        SOLFIL(2)=1
        SOLFIL(3)=NR0WS
        SOLFIL(4)=NCOL
        NRCOL=NR0WS+NCOL
        ICFFS=3*NRCCL
        INDEX=IOFFS+1
        DO 1 I=1,NRCOL
            J=I*3-2
            DATA(6)=0.
            DATA(7)=0.
C
C      READ ONE RECORD AT A TIME
C      IOFFS= THE LAST WORD USED BY NAMES AND FLAGS.
C      INDEX= NEXT AVAILABLE LOCATION TO STORE A NUMBER.
C
201    READ(IFLR,200,END=99) SOLIST(J), SOLIST(J+1),STAT,(DATA(L),L=1,5)
        IF(IFLGP.NE.0) READ(IFLR,201,END=99) DATA(6),DATA(7)
        FORMAT(E14.5,16X,E14.5)
        JW=J
        FORMAT(2A4,A2,5E14.5)
        K=I*6
        SCFLAG(K)=INDEX-IOFFS-1
C
C      FIND THE STATUS OF THE CURRENT RECORD.
        DO 2 J=1,15
            IF(STAT.NE.CSTAT(J)) GO TO 2
            ICOD=J-1
            GO TO 90
        CONTINUE
        IER=1001
        RETURN
        IHELP=ICOD
C
2      UPDATE THE 4 FIRST BITS.
        DO 3 J=1,4
            BITS(5-J)=.FALSE.
            IF(MOD(IHELP,2).EQ.1) BITS(5-J)=.TRUE.
            IHELP=IHELP/2
        CONTINUE
C
3

```

```

REAO0280
REAO0290
REAO0300
REAO0310
REAO0320
REAO0330
REAO0340
REAO0350
REAO0360
REAO0370
REAO0380
REAO0390
REAO0400
REAO0410
REAO0420
REAO0430
REAO0440
REAO0450
REAO0460
REAO0470
REAO0480
REAO0490
REAO0500
REAO0510
REAO0520
REAO0530
REAO0540
REAO0550
REAO0560
REAO0570
REAO0580
REAO0590
REAO0600
REAO0610
REAO0620
REAO0630
REAO0640
REAO0650
REAO0660
REAO0670
REAO0680
REAO0690
REAO0700
REAO0710
REAO0720
REAO0730
REAO0740
REAO0750

```



```

C      CHECK IF VALUES ARE ELIGIBLE FOR STORAGE AND
C      UPDATE THE CORRESPONDING BITS.
C
10    BITS(5) = .TRUE. 0.) GO TO 10
      IF(DATA(1).EQ.0.) GO TO 10
      BITS(5) = .FALSE.
      SOLIST(INDEX)=DATA(1)
      INDEX=INDEX+1
      BITS(6) = .TRUE.
      IF(DATA(2).EQ.0.) GO TO 20
      BITS(6) = .FALSE.
      SOLIST(INDEX)=DATA(2)
      INDEX=INDEX+1
      BITS(7) = .TRUE.
      IF(DATA(3).EQ.0.) GO TO 30
      IF(ICOD.EQ.2.JR.ICOD.EQ.4) GO TO 30
      IF(DATA(3).EQ.-TINF) GO TO 25
      BITS(7) = .FALSE.
      SOLIST(INDEX)=DATA(3)
      INDEX=INDEX+1
      GO TO 30
25    BITS(10) = .TRUE.
      BITS(8) = .TRUE.
      BITS(11) = .FALSE.
      IF(DATA(4).EQ.0.) GO TO 40
      IF(ICOD.EQ.3.JR.ICOD.EQ.4) GO TO 40
      IF(DATA(4).EQ.-TINF) GO TO 35
      BITS(8) = .FALSE.
      SOLIST(INDEX)=DATA(4)
      INDEX=INDEX+1
      GO TO 40
35    BITS(11) = .TRUE.
      BITS(9) = .TRUE.
      IF(DATA(5).EQ.0.) GO TO 50
      BITS(9) = .FALSE.
      SOLIST(INDEX)=DATA(5)
      INDEX=INDEX+1
40
C      CHECK THE STATUS OF PENALTIES.
C
      IHELP=10
      IF(DATA(6).EQ.0.) GO TO 60
      IHELP=30
      IF(DATA(6).EQ.-TINF) GO TO 60
      IHELP=20
      SOLIST(INDEX)=DATA(6)
50

```



```

SUBROUTINE READD IS EXACTLY AS THE SUBROUTINE READD
WITH THE ONLY DIFFERENCE THAT IT PACKS THE SOLUTION FILE
INTO THE SUPERSPARSE DATA STRUCTURE.
*****
SUBROUTINE READD(NROWS,NCOL,IER,IPLR,IPLW)
REAL DATA(7),SOLIST(1)
INTEGER*2 CSTAT(15),SOFLAG(1),STAT
LOGICAL*1 BITS(16)
COMMON /INIT/ZERO,TINF,IACI,MEMOR
EQUIVALENCE (SOLFIL(5),SOLIST(1),SOFLAG(1))
DATA CSTAT/2HIN,2HBS,2HLL,2HUL,2HEQ,2HVC,2HSC,2HFC,2HBC,
-2HVR,2HSR,2HRR,2HFR,2HER,2HTR/
IER=0
CALL TAPSET (2,IPLR,80,17,1,80)
READ(IPLR,100) NROWS,NCOL,IPLGP
FORMAT(15,30X,15,10X,11)
IEND=MEMOR-4
II=IEND
NRCOL=NROWS+NCOL
IOFFS=6*NRCOL
INDEX=IOFFS+1
DO 1 I=1,NRCOL
J=I*3-2
DATA(6)=0.
DATA(7)=0.
READ(IPLR,200) SOLIST(J),SOLIST(J+1),STAT,DATA(L),L=1,5)
IF(IPLGP.NE.0) READ(IPLR,201,END=99) DATA(6),DATA(7)
FORMAT(E14.5,16X,E14.5)
JW=J
DO 7 J=1,7
IF(DATA(J).EQ.0..OR.DATA(J).GE.TINF) GO TO 7
DO 8 K=1,IEND
IF(DATA(J).NE.SOLIST(K)) GO TO 8
DATA(K)=K
GO TO 7
CONTINUE
SOLIST(1)=DATA(J)
DATA(J)=1
II=II-1
CONTINUE
FORMAT(2A4,A2,5E14.5)
K=I*6
SOFLAG(K)=INDEX-IOFFS-1
DO 2 J=1,15

```

```

PEA00020
PEA00030
PEA00040
PEA00050
PEA00060
PEA00070
PEA00080
PEA00090
PEA00100
PEA00110
PEA00120
PEA00130
PEA00140
PEA00150
PEA00160
PEA00170
PEA00180
PEA00190
PEA00200
PEA00210
PEA00220
PEA00230
PEA00240
PEA00250
PEA00260
PEA00270
PEA00280
PEA00290
PEA00300
PEA00310
PEA00320
PEA00330
PEA00340
PEA00350
PEA00360
PEA00370
PEA00380
PEA00390
PEA00400
PEA00410
PEA00420
PEA00430
PEA00440
PEA00450
PEA00460
PEA00470
PEA00480
PEA00490

```



```

REA00500
REA00510
REA00520
REA00530
REA00540
REA00550
REA00560
REA00570
REA00580
REA00590
REA00600
REA00610
REA00620
REA00630
REA00640
REA00650
REA00660
REA00670
REA00680
REA00690
REA00700
REA00710
REA00720
REA00730
REA00740
REA00750
REA00760
REA00770
REA00780
REA00790
REA00800
REA00810
REA00820
REA00830
REA00840
REA00850
REA00860
REA00870
REA00880
REA00890
REA00900
REA00910
REA00920
REA00930
REA00940
REA00950
REA00960
REA00970

```

```

IF (STAT.NE.CSTAT(J)) GO TO 2
ICOD=J-1
GO TO 90
CONTINUE
IER=1001
RETURN
IHELP=ICOD
DO 3 J=1,4
  BITS(5-J)=.FALSE.
  IF (MOD(IHELP,2).EQ.1) BITS(5-J)=.TRUE.
  IHELP=IHELP/2
CONTINUE
  BITS(5)=.TRUE.
  IF (DATA(1).EQ.0.) GO TO 10
  BITS(5)=.FALSE.
  SOFLAG(INDEX)=IEND-DATA(1)
  INDEX=INDEX+1
  BITS(6)=.TRUE.
  IF (DATA(2).EQ.0.) GO TO 20
  BITS(6)=.FALSE.
  SOFLAG(INDEX)=IEND-DATA(2)
  INDEX=INDEX+1
  BITS(7)=.TRUE.
  IF (DATA(3).EQ.0.) GO TO 30
  IF (ICOD.EQ.2.OR.ICOD.EQ.4) GO TO 30
  IF (DATA(3).EQ.-1) GO TO 25
  BITS(7)=.FALSE.
  SOFLAG(INDEX)=IEND-DATA(3)
  INDEX=INDEX+1
  GO TO 30
  BITS(10)=.TRUE.
  BITS(8)=.TRUE.
  IF (DATA(4).EQ.0.) GO TO 40
  IF (ICOD.EQ.3.OR.ICOD.EQ.4) GO TO 40
  IF (DATA(4).EQ.-1) GO TO 35
  BITS(8)=.FALSE.
  SOFLAG(INDEX)=IEND-DATA(4)
  INDEX=INDEX+1
  GO TO 40
  BITS(11)=.TRUE.
  BITS(9)=.TRUE.
  IF (DATA(5).EQ.0.) GO TO 50
  BITS(9)=.FALSE.
  SOFLAG(INDEX)=IEND-DATA(5)
  INDEX=INDEX+1
  IHELP=10

```

REA00980  
 REA00990  
 REA01000  
 REA01010  
 REA01020  
 REA01030  
 REA01040  
 REA01050  
 REA01060  
 REA01070  
 REA01080  
 REA01090  
 REA01100  
 REA01110  
 REA01120  
 REA01130  
 REA01140  
 REA01150  
 REA01160  
 REA01170  
 REA01180  
 REA01190  
 REA01200  
 REA01210  
 REA01220  
 REA01230  
 REA01240  
 REA01250  
 REA01260  
 REA01270  
 REA01280  
 REA01290  
 REA01300  
 REA01310  
 REA01320  
 REA01330  
 REA01340  
 REA01350  
 REA01360  
 REA01370  
 REA01380  
 REA01390  
 REA01400  
 REA01410  
 REA01420  
 REA01430  
 REA01440  
 REA01450

```

IF(DATA(6).EQ.0.) GO TO 60
IHELP=30
IF(DATA(6).EQ.TINF) GO TO 60
IHELP=20
SFLAG(INDEX)=IEND-DATA(6)
INDEX=INDEX+1
IHELP=1
IF(DATA(7).EQ.0.) GO TO 70
IHELP=3
IF(DATA(7).EQ.TINF) GO TO 70
IHELP=2
SFLAG(INDEX)=IEND-DATA(7)
INDEX=INDEX+1
IHELP=IHELP+IHELP1
ICOD=IHELP-11
IF(IHELP.LE.13) GO TO 80
ICOD=IHELP-18
IF(IHELP.LE.23) GO TO 80
ICOD=IHELP-25
DO 4 J=1,4
  BITS(16-J)=.FALSE.
IF(MOD(ICOD,2).EQ.1) BITS(16-J)=.TRUE.
ICOD=ICOD/2
CONTINUE
BITS(16)=.FALSE.
IHELP=0
DO 5 J=1,16
  IF(BITS(17-J)) IHELP=IHELP+2**((J-1)
CONTINUE
SFLAG(K-1)=IHELP
CONTINUE
IF(MOD(INDEX,2).NE.0) INDEX=INDEX-1
INDEX=INDEX/2
DO 11 J=1,IEND
  INDEX=INDEX+1
  SOLIST(INDEX)=SOLIST(J)
CONTINUE
INDEX=INDEX+4
WRITE(6,300) INDEX
FORMAT(5X,'WORDS REQUIRED FOR PACKED FILE: ',I6)
SOLFIL(1)=INDEX
SOLFIL(2)=0
SOLFIL(3)=NROWS
SOLFIL(4)=NCOL
WRITE(1,FLW,1) (SOLFIL(I),I=1,INDEX)
RETURN
WRITE(6,999) SOLIST(JW),SOLIST(JW+1),STAT,(DATA(L),L=1,5)
FORMAT(2X,2A4,A2,5F14.5)
  
```

```

IER=1002
RETURN
END
*****
SUBROUTINE INTERP EXECUTES THE QUERY USING THE CODE
GENERATED BY THE PARSE SUBROUTINE.
CODE - THE CODE GENERATED BY THE PARSE
MASK - THE MASK FIELD OF THE QUERY
ARG2, ARG4 - THE VALUES OF ARG1, ARG2 OF CONDITION FIELD
K1, K2, K3 - AS THEY ARE SET BY THE SET CMMAND
K1=START K2=END K3=STEP
*****
SUBROUTINE INTERP(CODE,MASK,ARG2,ARG4,K1,K2,K3,NROWS,NCOL,BLANK,ST
-AR,IER,IPLG)
COMMON /INIT/ ZERO,IINF,IAC
INTEGER #2 BLANK,STAR,CODE(20),NAME(8),MASK(8),SNAME(8),SOFLAG(1)
COMMON /SOLPAC/ SOLFIL(81920)
EQUIVALENCE (SOLFIL(5),SOLIST(1),SOFLAG(1))
INTEGER PROP(7)
REAL SOLIST(1),RECORD(7),RECAD(7)
LOGICAL EVCOD,EVMN
IER=0
SET UP INFORMATIONS FOR PRINT RESULTS.
INITIALIZE VALUES FOR MINIMUM AND MAXIMUM.
IPRINT=2
SMIN=.7273E74
SMAX=-SMIN
DO 4 J=1,7
RECAD(J)=0.
PROP(J)=0.
IF(CODE(11).EQ.0) PROP(J)=J
CONTINUE
IF(CODE(11).EQ.0) GO TO 12
DO 5 I=1,7
JSAVE=0
DO 6 J=12,18
IF(CODE(J).NE.I) GO TO 6
JSAVE=J
GO TO 11
CONTINUE
IF(JSAVE.EQ.0) GO TO 12
PROP(I)=JSAVE-11
CONTINUE

```

```

REA01460
REA01470
REA01480
INT00010
INT00020
INT00030
INT00040
INT00050
INT00060
INT00070
INT00080
INT00090
INT00100
INT00110
INT00120
INT00130
INT00140
INT00150
INT00160
INT00170
INT00180
INT00190
INT00200
INT00210
INT00220
INT00230
INT00240
INT00250
INT00260
INT00270
INT00280
INT00290
INT00300
INT00310
INT00320
INT00330
INT00340
INT00350
INT00360
INT00370
INT00380
INT00390
INT00400
INT00410
INT00420
INT00430
INT00440
INT00450

```



```

C      SET UP LIMITS FOR SEARCHING.
C12
C      NEND=K2
C      NCUR=K1
C      IF(CODE(2).EQ.3.AND.K2.GT.NROWS) NEND=NROWS
C      IF(CODE(2).NE.2) GO TO 91
C      NCUR=NROWS+K3+K1-1
C      NEND=NROWS+K2
C      IF(NEND.GT.(NROWS+NCOL)) NEND=NROWS+NCOL
C      IF(CODE(2).EQ.3) IPRINT=3
C      NCUR=NCUR-K3
C      ICOUNT=0
C      NCUR=NCUR+K3
C      IF(NCUR.LE.NEND) GO TO 10
C
C      IF END OF SEARCHING PRINT BOTTOM LINE.
C
C      IF(CODE(1).NE.17.OR.CODE(2).NE.1) GO TO 105
C      IACT=0
C      WRITE(6,500)
C      FORMAT(10X,' FILE DEACTIVE ',//)
C      RETURN
C500
C      IF(CODE(4).NE.9.AND.CODE(5).GT.6)CALL PRRES(CODE, SNAME, RECAD, ICODEM
C      -X, ICOUNT, PROP)
C105
C      CALL PRBOT(CODE, RECORD, RECAD, ICOUNT, MASK, PRCP, IPRINT)
C      RETURN
C      IF(IACT.EQ.0) GO TO 102
C
C      CHECK IF ACTIVE FILE EXISTS.
C      IF IT IS CHECK IF CURRENT RECORD ACTIVE.
C      IF NOT GET THE NEXT RECORD IF ANY.
C
C      IHLP=SOFLAG(NCUR*6-1)
C      IF(MOD(IHLP,2).EQ.0.AND.CODE(1).NE.16) GO TO 99
C      IF(MOD(IHLP,2).EQ.1.AND.CODE(1).EQ.16) GO TO 99
C      IF(CODE(1).NE.17.OR.CODE(2).NE.1) GO TO 102
C      SOFLAG(NCUR*6-1)=SOFLAG(NCUR*6-1)-1
C      GO TO 99
C
C      PUT BLANKS TO THE NAME.
C
C      DO 2 J=1,8
C102
C      NAME(J)=BLANK
C      CONTINUE
C      GET THE NAME OF CURRENT RECORD.
C
C      CALL CURNAM(MASK, NAME, NCUR, BLANK, STAR)

```

```

C      CHECK IF NAME MATCHES TO THE MASK.
C      IF NOT GET THE NEXT RECORD.
C      DO 1 I=1,8
C      IF (MASK(I).EQ.STAR) GO TO 1
C      IF (MASK(I).NE.NAME(I)) GO TO 99
C      CONTINUE
C      IF IFLG=0 (SPARSE STRUCTURE) CALL DEPACK TO GET
C      THE VALUES OF CURRENT RECORD ELSE CALL DPAC.
C      IF (IFLG.EQ.0) GO TO 201
C      CALL DEPACK(CODE, ICOD, RECORD, NCUR, BLANK, NROWS, NCOL, IER)
C      GO TO 202
C      CALL DPAC(CODE, ICOD, RECORD, NCUR, BLANK, NROWS, NCOL, IER)
C      IF (IER.NE.0) RETURN
C      IF NO CONDITION FIELD SKIP
C      ELSE EVALUATE CONDITION USING EVCOD AND EVMN.
C      IF (CODE(4).EQ.0) GO TO 60
C      IF (CODE(4).EQ.9) GO TO 50
C      IF (CODE(5).LT.7) GO TO 21
C      IF (CODE(5).NE.7) GO TO 22
C      EVALUATE BOUND CONDITION. (MAX OR MIN)
C      IF (RECORD(CODE(4)).GT.SMIN) GO TO 99
C      IF (RECORD(CODE(4)).EQ.SMIN) GO TO 24
C      $MIN=RECORD(CODE(4))
C      GO TO 23
C      IF (RECORD(CODE(4)).LT.SMAX) GO TO 99
C      IF (RECORD(CODE(4)).EQ.SMAX) GO TO 24
C      $MAX=RECORD(CODE(4))
C      ICOUNT=0
C      DO 7 I=1,7
C      RECAD(I)=RECORD(I)
C      $NAME(I)=NAME(I)
C      CONTINUE
C      $NAME(8)=NAME(8)
C      ICODMX=ICOD
C      ICOUNT=ICOUNT+1
C      GO TO 99
C      IF (EVCOD(CODE, ICOD, RECORD, ARG2, ARG4, IER)) GO TO 60
C      GO TO 99
C      INDEX=5
C      IF (.NOT. EVMN(CODE, ICOD, RECORD, INDEX, MIN, MAX, ICOUNT, IER)) GO TO 99

```

```

INT00940
INT00950
INT00960
INT00970
INT00980
INT00990
INT01000
INT01010
INT01020
INT01030
INT01040
INT01050
INT01060
INT01070
INT01080
INT01090
INT01100
INT01110
INT01120
INT01130
INT01140
INT01150
INT01160
INT01170
INT01180
INT01190
INT01200
INT01210
INT01220
INT01230
INT01240
INT01250
INT01260
INT01270
INT01280
INT01290
INT01300
INT01310
INT01320
INT01330
INT01340
INT01350
INT01360
INT01370
INT01380
INT01390
INT01400
INT01410

```







```

C C C
* SUBROUTINE CURNAM RETURNS THE NAME OF THE CURRENT
* RECORD AT THE ARRAY 'NAME'.
* *****
SUBROUTINE CURNAM(MASK,NAME,NCUR,BLANK,STAR)
COMMON /SOLPAC/SOLFIL(81920)
REAL PACK(1)
LOGICAL *1 NAME(16),LOGPAC(1)
INTEGER *2 MASK(8),BLANK,STAR
EQUIVALENCE (SOLFIL(5),PACK(1),LOGPAC(1))
INDEX=NCUR*12-12
DO 1 I=2,16,2
J=1/2
NAME(I-1)=LOGPAC(INDEX+J)
CONTINUE
RETURN
END
C C C C C C C
* SUBROUTINE DEPACK RETURNS AT ARRAY RECCD EXPLICITLY
* ALL THE VALUES OF THE CURRENT RECORD AND THE STATUS AT ICOD
* IT IS USED TO GET THE VALUES FROM THE SPARSE DATA STRUCTURE.
* *****
SUBROUTINE DEPACK(CODE,ICOD,RECORD,NCUR,BLANK,NROWS,NCOL,IER)
REAL RECCD(7),SOLIST(1)
INTEGER *2 CODE(20),SOFLAG(1)
LOGICAL *1 BITS(16),SONAME(1)
COMMON /INIT/ZERO,TINF,IACT
COMMON /SOLPAC/SOLFIL(81920)
EQUIVALENCE (SOLFIL(5),SOLIST(1),SOFLAG(1),SONAME(1))
IER=0
C C C
CREATE THE BIT PATTERN.
IOFFS=3*(NROWS+NCOL)+1
IBITS=NCUR*6-1
IHELP=SOFLAG(1BITS)
C C C
IF BITS NEGATIVE MAKE IT POSITIVE
IF (IHELP.LT.0) IHELP=IHELP+65536
DO 1 I=1,16
BITS(I7-1)=.FALSE.
IF (MOD(IHELP,2).EQ.1) BITS(I7-1)=.TRUE.
IHELP=IHELP/2
CONTINUE
C C C
C C C C C C C
CUR00030
CUR00040
CUR00050
CUR00060
CUR00070
CUR00080
CUR00090
CUR00100
CUR00110
CUR00120
CUR00130
CUR00140
CUR00150
CUR00160
CUR00170
CUR00180
CUR00190
DEP00010
DEP00020
DEP00030
DEP00040
DEP00050
DEP00060
DEP00070
DEP00080
DEP00090
DEP00100
DEP00110
DEP00120
DEP00130
DEP00140
DEP00150
DEP00160
DEP00170
DEP00180
DEP00190
DEP00200
DEP00210
DEP00220
DEP00230
DEP00240
DEP00250
DEP00260
DEP00270
DEP00280
DEP00290
DEP00300
DEP00310

```

DEP00320  
DEP00330  
DEP00340  
DEP00350  
DEP00360  
DEP00370  
DEP00380  
DEP00390  
DEP00400  
DEP00410  
DEP00420  
DEP00430  
DEP00440  
DEP00450  
DEP00460  
DEP00470  
DEP00480  
DEP00490  
DEP00500  
DEP00510  
DEP00520  
DEP00530  
DEP00540  
DEP00550  
DEP00560  
DEP00570  
DEP00580  
DEP00590  
DEP00600  
DEP00610  
DEP00620  
DEP00630  
DEP00640  
DEP00650  
DEP00660  
DEP00670  
DPA00010  
DPA00020  
DPA00030  
DPA00040  
DPA00050  
DPA00060  
DPA00070  
DPA00080  
DPA00090  
DPA00100  
DPA00110  
DPA00120

```

IHELP=0
ICODP=0
DO 2 I=1,4 IHELP=IHELP+2*(I-1)
IF(BITS(5-I)) ICODP=ICODP+2*(I-1)
CONTINUE
ICOD=IHELP
INDEX=IOFFS+SOFLAG(1BITS+1)
IF(SOFLAG(1BITS+1).LT.0) INDEX=INDEX+65536
DO 3 I=1,5
RECORD(I)=0
IF(I.EQ.3.AND.(ICOD.EQ.2.OR.ICOD.EQ.4)) GO TO 10
IF(I.EQ.4.AND.(ICOD.EQ.3.OR.ICOD.EQ.4)) GO TO 10
IF(BITS(4+I)) GO TO 3
RECORD(I)=SOLIST(INDEX)
INDEX=INDEX+1
GO TO 3
RECORD(1)=RECORD(1)
CONTINUE
IF(BITS(10)) RECORD(3)=-TINF
IF(BITS(11)) RECORD(4)=TINF
RECORD(6)=0
IF(ICODP.LE.2) GO TO 20
IF(ICODP.GT.5) GO TO 30
RECORD(6)=SOLIST(INDEX)
INDEX=INDEX+1
GO TO 20
RECORD(6)=TINF
RECORD(7)=0
IF(ICODP.EQ.0.OR.ICODP.EQ.3.OR.ICODP.EQ.6) RETURN 40
IF(ICODP.EQ.2.OR.ICODP.EQ.5.OR.ICODP.EQ.8) GO TO 40
RECORD(7)=SCLIST(INDEX)
RETURN
RECORD(7)=TINF
RETURN
END
*****
* SUBROUTINE DPAC RETURNS THE SAME VALUES AS DEPACK
* BUT USING THE SUPER SPARSE DATA STRUCTURE.
*****
SUBROUTINE DPAC(CODE, ICOD, RECORD, NCUR, BLANK, NROWS, NCOL, IER)
REAL RECORD(7), SOLIST(1)
INTEGER*2 CODE(20), SOFLAG(1)
LOGICAL*1 BITS(16), SONAME(1)
COMMON /INIT/ZERO, TINF, IACT, MEMOR, ISIZE
COMMON /SOLPAC/SOLFIL(81920)

```

2

10  
3

30  
20

40

C  
C  
C  
C  
C



```

EQUIVALENCE (SOLF IL(5), SOLIST(1), SOFLAG(1), SONAME(1))
IER=0
IOFFS=6*(NRQWS+NCOL)+1
IEND=ISIZE-4
IBITS=NCUR*6-1
IHELP=SOFLAG(1)
IF(IHELP.LT.0) IHELP=IHELP+65536
DO I=1,16
  BITS(17-I)=.FALSE.
  IF(MOD(IHELP,2).EQ.1) BITS(17-I)=.TRUE.
  IHELP=IHELP/2
CONTINUE
IHELP=0
ICODP=0
DO I=1,4
  IHELP=IHELP+2**((I-1))
  IF(BITS(16-I)) ICODP=ICODP+2**((I-1))
CONTINUE
ICOD=IHELP
INDEX=IOFFS+SOFLAG(1)
IF(SOFLAG(1).LT.0) INDEX=INDEX+65536
DO I=1,5
  RECORD(I)=0
  IF(I.EQ.3.AND.(ICOD.EQ.2.OR.ICOD.EQ.4)) GO TO 10
  IF(I.EQ.4.AND.(ICOD.EQ.3.OR.ICOD.EQ.4)) GO TO 10
  IF(BITS(4+I)) GO TO 3
  RECORD(I)=SOLIST(IEND-SOFLAG(INDEX))
  INDEX=INDEX+1
  GO TO 3
RECORD(1)=RECORD(1)
CONTINUE
IF(BITS(10)) RECORD(3)=.T INF
IF(BITS(11)) RECORD(4)=.T INF
RECORD(6)=0
IF(ICODP.LE.2) GO TO 20
IF(ICODP.GT.5) GO TO 30
RECORD(6)=SOLIST(IEND-SOFLAG(INDEX))
INDEX=INDEX+1
GO TO 20
RECORD(6)=.T INF
RECORD(7)=0
IF(ICODP.EQ.0.OR.ICODP.EQ.3.OR.ICODP.EQ.6) RETURN
IF(ICODP.EQ.2.OR.ICODP.EQ.5.OR.ICODP.EQ.8) GO TO 40
RECORD(7)=SOLIST(IEND-SOFLAG(INDEX))
RETURN
RECORD(7)=.T INF
RETURN
END

```

```

DPA00130
DPA00140
DPA00150
DPA00160
DPA00170
DPA00180
DPA00190
DPA00200
DPA00210
DPA00220
DPA00230
DPA00240
DPA00250
DPA00260
DPA00270
DPA00280
DPA00290
DPA00300
DPA00310
DPA00320
DPA00330
DPA00340
DPA00350
DPA00360
DPA00370
DPA00380
DPA00390
DPA00400
DPA00410
DPA00420
DPA00430
DPA00440
DPA00450
DPA00460
DPA00470
DPA00480
DPA00490
DPA00500
DPA00510
DPA00520
DPA00530
DPA00540
DPA00550
DPA00560
DPA00570
DPA00580
DPA00590
DPA00600

```













```

11 CONTINUE
10 WRITE(6,200)
200 FORMAT(1H)
    RETURN
END
*****
** SUBROUTINE PRBOT PRINTS THE TOTAL NUMBER OF QUALIFIED
** RECORDS AFTER THE LAST RECORD PRINTED BY PRRES SUBROUTINE.
** ALSO IT PRINTS THE RESULTS FROM ADD AND AVERAGE COMMANDS.
**
*****
SUBROUTINE PRBOT(CCODE,RECORD,RECAD,ICOUNT,MASK,PRGP,IPRINT)
COMMON /IN IT/ZERO,TINF,IACT
INTEGER*2 CODE(20),MASK(8)
INTEGER PROP(7)
REAL RECORD(7),RECAD(7)
IF(CODE(1).GT.15) GO TO 30
*****
IF ADD OR AVERAGE COMMANDS SKIP TO 10
IF(CODE(1).EQ.13.OR.CODE(1).EQ.14) GO TO 10
FORMAT(2X,7F10.3)
WRITE(6,100) ICOUNT,(MASK(I),I=1,8)
FORMAT(2X,/,2X,15,2X,'ROWS OR COLUMNS WITH MASK : ',8A1,/,2X,'SATI
-SEY THE CONDITIONS')
RETURN
IF(CODE(1).EQ.13) GO TO 20
DO 1 J=1,7
RECAD(J)=RECAD(J)/ICOUNT
CONTINUE
PROP(1)=1
PROP(2)=2
PROP(3)=5
PROP(4)=0
CALL PRHEAD(CODE,PROP,IPRINT)
CALL PRRES(CODE,MASK,RECAD,ICOD,ICOUNT,PROP)
GO TO 21
IF(CODE(1).EQ.16) GO TO 31
IACT=IACT-ICOUNT
WRITE(6,300) ICOUNT,IACT
FORMAT(5X,/,5X,16,' RECORDS DEACTIVATED',/,5X,' TOTAL RECORDS
- ACTIVE :',16)
RETURN
IACT=IACT+ICOUNT
WRITE(6,400) ICOUNT,IACT
FORMAT(5X,/,5X,16,' RECORDS ACTIVATED',/,5X,' TOTAL RECORDS ACTI

```

AD-A070 093

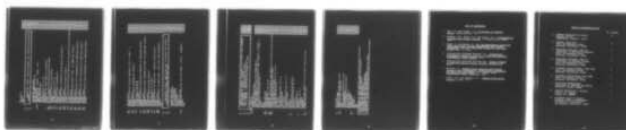
NAVAL POSTGRADUATE SCHOOL MONTEREY CA  
ATHENA: A SYSTEM TO INTERACTIVELY ANALYZE LARGE SCALE OPTIMIZAT--ETC(U)  
MAR 79 P I GALATAS

F/G 9/2

UNCLASSIFIED

NL

20F2  
AD  
A070093



END  
DATE  
FILMED

7-79  
DDC



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A





```

56 WRITE(6,505)
505 FORMAT(5X,'INVALID LOGICAL OPERATOR ''OR'', ''AND'', '' )
RETURN
57 WRITE(6,506)
506 FORMAT(5X,'INVALID OPERAND FOR STATUS')
RETURN
600 IF(IHELP.EQ.2) GO TO 54
IF(IHELP.EQ.1) GO TO 61
WRITE(6,601)
601 FORMAT(5X,'INVALID RELATIONAL OPERATOR IN CONDITION FIELD')
RETURN
61 WRITE(6,602)
602 FORMAT(5X,'INVALID FIRST OPERAND FOR RELOP IN CONDITION FIELD')
RETURN
700 WRITE(6,701)
701 FORMAT(5X,'INVALID PRINT FIELD')
RETURN
999 IF(IER.NE.1001) GO TO 900
WRITE(6,801)
801 FORMAT(5X,'ERROR IN INPUT DATA - INVALID STATUS CODE')
RETURN
900 WRITE(6,901)
901 FCRMAT(5X,'ERROR IN INPUT DATA - DATA LESS THAN REQUIRED')
RETURN
END
*****
C ***** SUBROUTINE TIMER PRINTS THE TIME IN SECONDS THE VIRTUAL *****
C ***** MACHINE WAS USED SINCE THE LAST CALL. *****
C ***** *****
C *****
C ***** SUBROUTINE TIMER *****
COMMON ITM(6),TIME
CALL IXCLOCK(ITM)
SEC=ITM(5)/76800.
EL=SEC-TIME
TIME=SEC
WRITE(6,100) EL
100 FORMAT(10X,'EXECUTION TIME FOR QUERY =',F8.4,' SECONDS')
RETURN
END

```

```

ERR00460
ERR00470
ERR00480
ERR00490
ERR00500
ERR00510
ERR00520
ERR00530
ERR00540
ERR00550
ERR00560
ERR00570
ERR00580
ERR00590
ERR00600
ERR00610
ERR00620
ERR00630
ERR00640
ERR00650
ERR00660
ERR00670
ERR00680
ERR00690
ERR00700
TIM00010
TIM00020
TIM00030
TIM00040
TIM00050
TIM00060
TIM00070
TIM00080
TIM00090
TIM00100
TIM00110
TIM00120
TIM00130
TIM00140
TIM00150
TIM00160

```







REW00470  
 REW00480  
 REW00490  
 REW00500  
 REW00510  
 REW00520  
 REW00530  
 REW00540  
 REW00550  
 REW00560  
 REW00570  
 REW00580  
 REW00590  
 REW00600  
 REW00610  
 REW00620

```

99 GC TO 30
    NCOL=MET-NROWS
    WRITE(2,1,600) NROWS,NCOL,IFLGP
600 FORMAT(15,30X,15,10X,11)
    IA=1
    READ(2,1A,600) NROWS,NCOL,IFLGP
    ITOT=NROWS+NCOL
    WRITE(9,600) NROWS,NCOL,IFLGP
    DO 6 I=1,ITOT
700 READ(2,1A,700) A
    WRITE(9,700) A
    FORMAT(20A4)
    CONTINUE
    END FILE 9
    STOP
    END
    //GO.FT01F001 DD DISP=OLD,UNIT=3400-4,VOL=SER=(NPS251,NPS631),
    //DSNAME=MP$LIST,DCB=(RECFM=UA,BLKSIZE=133,DEN=2)
    //GO.FT02F001 DD DSN=SI951,STADFM,
    //UNIT=3330,VOL=SER=DISK04,
    //SPACE=(CYL,(6,1)),DISP=(NEW,KEEP),
    //DCB=(RECFM=FB,LRECL=80,BLKSIZE=6400)
    //GO.FT09F001 DD UNIT=2400,VOL=SER=NPS713,DISP=(,KEEP),
    //LABEL=(,SL),DSN=TAPFOR,DCB=(RECFM=FB,LRECL=80,BLKSIZE=6400,DEN=2)
  
```

### LIST OF REFERENCES

1. Aho, A.V. and Ullman, J.D., Principles of Compiler Design, Addison-Wesley, 1977.
2. Bradley, G.H., Brown, G.G. and Graves, G.W., Preprocessing of Large Scale Linear Programs (in preparation), March 1979.
3. Brown, G. and Graves, G., XS - An Experimental System for LARGE SCALE MIXED INTEGER OPTIMIZATION with Elastic Programming, National ORSA/TIMS convention, Las Vegas, Nevada, Nov. 17, 1975.
4. International Business Machines Inc., Mathematical Programming System/360 Version 2, Linear and Separable Programming - User's Manual, 1971.
5. International Business Machines Inc., Control Program 67/Cambridge Monitor System (CP/CMS), Version 3.25, 1974.
6. Mavrikas, C., Optimal 5-year Planning Using Mixed-Integer Linear Programming. Three Models Implemented for Naval Air Test Center, M.S. Thesis, Naval Postgraduate School, Monterey, 1979.
7. O'Neil, R.P. and Sanders, R.C., PERUSE System Manual Version 2, Sept. 1977.



INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 55 Department of Operations Research Naval Postgraduate School Monterey, California 93940	2
4. Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
5. Professor Gordon Bradley, Code 52BZ Naval Postgraduate School Monterey, California 93940	2
6. Professor Gerald Brown, Code 55BW Naval Postgraduate School Monterey, California 93940	1
7. Greek Army Headquarters Department of Research and Design Holargos, Athens, GREECE	2
8. Captain Panagiotis I. Galatas Alketou 19, Pagrati Athens 506, GREECE	3
9. Professor Glenn W. Graves Graduate School of Management University of California Los Angeles, California 90024	1